DOCUMENT RESUME

ED 376 798 IR 016 898

AUTHOR	Zupancic, Joze, Ed.; Wrycza, Stanislaw, Ed.
TITLE	Information Systems Development--ISD '94. Methods & Tools. Theory & Practice. Proceedings of the International Conference (4th, Bled, Slovenia, September 20-22, 1994).
INSTITUTION	Gdansk Univ. (Poland).; Maribor Univ. (Slovenia).
REPORT NO	ISBN-86-81049-78-X
PUB DATE	Sep 94
NOTE	734p.
PUB TYPE	Books (010) -- Collected Works - Conference Proceedings (021)
EDRS PRICE	MF04/PC30 Plus Postage.
DESCRIPTORS	Case Studies; *Computer Software Development; *Engineering; Foreign Countries; Information Networks; *Information Systems; Information Technology; Models; *Organizational Development; Planning; Research Reports; *Systems Development; *Technological Advancement

ABSTRACT

These proceedings present 3 invited papers, 65 submitted papers, and 17 presentations on work in progress that were given at the Fourth International Conference on Information Systems Development. The three invited papers are: "Information Systems Planning in Small Business" (Georgios Doukidis, Panagiotis Lybereas, Robert D. Galliers); "Development of Information Systems To Support Electronic Commerce" (Joze Gricar); and "The Evolution of the Information Systems Field: The Impact of Technology on the Teaching and Practice of Information Systems Development" (A. Milton Jenkins). The remaining papers are grouped into the following areas: (1) modelling; (2) system development; (3) organizational aspects; (4) strategy planning; (5) applications; (6) reengineering; (7) case tools; (8) specific aspects of information engineering; (9) quality; (10) requirements of engineering and system specification; (11) education; (12) special aspects and comparisons; (13) computer-supported cooperative work; (14) technical aspects of information systems design; and (15) reports on work in progress. (SLD)

****
* Reproductions supplied by EDRS are the best that can be made *
* from the original document. *
****

ED 376 798

I

ISD '94
International conference

U.S. DEPARTMENT OF EDUCATION
Office of Educational Research and Improvement
EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)

☐ This document has been reproduced as received from the person or organization originating it.

☐ Minor changes have been made to improve reproduction quality.

● Points of view or opinions stated in this document do not necessarily represent official OERI position or policy

Edited by
Jože Zupančič and Stanislaw Wrycza

Proceedings of The Fourth International Conference

INFORMATION SYSTEMS DEVELOPMENT - ISD '94

Methods & Tools
Theory & Practice

"PERMISSION TO REPRODUCE THIS MATERIAL HAS BEEN GRANTED BY
Joze Zupancic

TO THE EDUCATIONAL RESOURCES INFORMATION CENTER (ERIC)."

BEST COPY AVAILABLE

IR 016898

Bled, Slovenia, 20-22 September 1994

Proceedings of
The Fourth International
Conference
INFORMATION SYSTEMS
DEVELOPMENT - ISD'94

Methods & Tools
Theory & Practice

Bled, Slovenia, 20-22 September 1994

Edited by

Jože Zupančič and Stanislaw Wrycza

Organized by

University of Maribor, School of Organizational Sciences
&
University of Gdansk, Department of Information Systems

Title:
Proceedings of The Fourth International Conference
INFORMATION SYSTEMS DEVELOPMENT- ISD'94
Methods & Tools, Theory & Practice

Bled (Slovenia), 20-22 September 1994

Edited by
Jože Zupančič and Stanislaw Wrycza

Published by
Moderna Organizacija, Kranj

Printed by
KAVC d.o.o., Kranj

Edition size: 220

ORGANIZING COMMITTEE
Cene Bavec, Ministry of Science and Technology, Ljubljana
Meta Benčič, Protokolarni servis Brdo
Božidar Blatnik, SRC d.o.o, Ljubljana
Jože Florjančič, University of Maribor, School of Organizational Science, *Chair*
Nace Pavlin, University of Maribor, School of Organizational Science
Stanislaw Wrycza, University of Gdansk, Department of Information Systems
Jože Zmrzlikar, Iskra - Tovarna sestavnih delov, Kranj (Iskra - Component Factory)
Jože Zupančič, University of Maribor, School of Organizational Science

CIP - Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica

061.3(497.12 Bled):659.2

INTERNATIONAL Conference Information Systems Development - ISD'94, (4 ; 1994 ; Bled)

Methods & Tools, Theory & Practice : proceedings of the Fourth International Conference Information Systems Development - ISD'94, Bled (Slovenia), 20-22 September 1994 / edited by Jože Zupančič and Stanislaw Wrycza; [organized by] University of Maribor, School of Organizational Science & University of Gdansk, Department of Information Systems. - Kranj : Moderna organizacija, 1994

ISBN 86-81049-78-X

1. Gl. stv. nasl. 2. Zupančič, Jože 3. Wrycza, Stanislaw
42201600

Po mnenju Ministrstva za šolstvo in šport št. 415-8/94 dne 29.8.94 šteje ta knjiga po 13. točki tarifne številke 3 Tarife davka od prometa proizvodov in storitev (Ur. list RS, št. 4/92) med proizvode, za katere se plačuje 5% davka od prometa proizvodov in storitev.

INTERNATIONAL PROGRAM COMMITTEE

Gary Allen, University of Huddersfield (UK)
David Avison, University of Southampton (UK)
Christopher Barry, University College Galway (Ireland)
Niels Bjorn-Andersen, Copenhagen Business School, (Denmark)
Cornelia Boldyreff, University of Durham (UK)
Saša Dekleva, DePaul University, Chicago, USA
Oscar Diaz, Universitad del Pais Vasco (Spain)
E.N. El-Sayed, University of Alexandria (Egypt)
Eckhard Falkenberg, University of Nijmegen (The Netherlands)
Guy Fitzgerald, Birkbeck College, London (UK)
Victor Gladun, Institute for Applied Informatics, Kiev (Ukraine)
Edwin Gray, Glasgow Caledonian University (UK)
Igor Hawryszkiewycz, University of Technology Sydney, (Australia)
Juhani Iivari, University of Jyvaskyla (Finland)
Janete Ilmete, University of Riga (Latvia)
Jerzy Kisielnicki, University of Warsaw (Poland)
Rumjana Kirkova, Bulgarian Academy of Sciences, Sofia (Bulgaria)
Marjan Krisper, University of Ljubljana (Slovenia)
Winfried Lamersdorf, University of Hamburg (Germany)
Roland Langerhorst, Cap Volmac B.V. (The Netherlands)
Leszek Maciaszek, Macquarie University (Australia)
Ir. Rik Maes, University of Amsterdam (The Netherlands)
Heinrich Mayr, University Klagenfurt (Austria)
Anders Nilsson, Stockholm School of Economics (Sweden)
Eugene Ovsyannikov, The Academy of Sciences of Russia, Sankt Petersburg (Russia)
Ivan Rozman, University of Maribor (Slovenia)
Bernd Schiemenz, Philipps-University Marburg (Germany)
Stefano Spaccapietra, Laboratoire de bases de donnees, Lausanne (Switzerland)
Roland Stamper, University Twente, Enschede, (The Netherlands)
Frank Stowell, University of Paisley (UK)
Bo Sundgren, Statistics Sweden, Stockholm (Sweden)
Veljko Topolovec, University of Zagreb (Croatia)
Roland Traunmueller, University Linz (Austria)
Tadeusz Wierzbicki, University of Szczecin (Poland)

CO-CHAIRMEN
Jože Zupančič, University of Maribor (Slovenia)
Stanislaw Wrycza, University of Gdansk (Poland)

TECHNICAL COMMITTEE

Nace Pavlin, Stanka Heberle, Belinda Škofic, Branislav Šmitek, Borut Verber, Tatjana Žerovnik

In addition to the help of the members of the International Programme Committee, we also appreciate the kind assistance from the following colleagues who helped us review the conference papers submitted to ISD'94:

Martin Andersson, Ecole polytechnique federale de Lausane, Laboratoire de bases de donnee, Switzerland
Robert L. Ashenhurst, University of Chicago, USA
Annamaria Auddino, Ecole polytechnique federale de Lausane, Laboratoire de bases de donnees, Switzerland
Marko Bohanec, Institute Jožef Stefan, Ljubljana
Angele Cavaye, Copenhagen Business School, Denmark
Malcolm K. Crowe, University of Paisley, Paisley, United Kingdom
Marti Dunlop, University of Paisley, Paisley, United Kingdom
Edi Fontana, Ecole polytechnique federale de Lausane, Laboratoire de bases de donnees, Switzerland
Miro Gradišar, University of Maribor, Slovenia
Uldis Grivinš, University of Latvia, Riga, Latvia
D.S. Hinley, University of Durham, United Kingdom
James G. Howell, University of Paisley, Paisley, United Kingdom
Janez Jereb, University of Maribor, Slovenia
Jože Jesenko, University of Maribor, Slovenia
Howard A. Kanter, DePaul University, Chicago, USA
J.W. Kawala, University of Paisley, Paisley, United Kingdom
Martin Kolkman, Van der Zwan Consultants, Den Haag, The Netherlands
Kari Kuutti, University of Oulu, Finland
Robert Leskovar, University of Maribor, Slovenia
Ying Liang, University of Paisley, Paisley, United Kingdom
Kecheng Liu, Staffordshire University, United Kingdom
Anton Ljubič, University of Maribor, Slovenia
Tomaž Mohorič, University of Ljubljana, Slovenia
Angus Quin, University of Paisley, Paisley, United Kingdom
Vladislav Rajkovič, University of Maribor, Slovenia
D. Robson, University of Durham, United Kingdom
Linda Salchenberger, Loyola University Chicago, USA
Cees van Slooten University of Twente, Enschede, The Netherlands
M.H. Stansfield, University of Paisley, Paisley, United Kingdom
James S. Weber, The University of Illinois at Chicago, USA
Conrad Weisert, 1205 Madison Park, Chicago, USA
Daune West, University of Paisley, Paisley, United Kingdom
Fons Wijnhoven, University of Twente, Enschede, The Netherlands

CONTENTS

Preface 1

Information Systems Planning in Small Business, *Georgios Doukidis, Panagiotis Lybereas, Robert D. Galliers (Invited Paper)* 3

Development of Information Systems to Support Electronic Commerce, *Jože Gričar (Invited paper)* 19

MODELLING

Analyzing the Notions of Attribute, Aggregate, Part, and Member in Data/ Knowledge Modelling, *Renate Motschnig-Pitrik* 31

Entity-Relationship Modelling of Information System with Deductive Capabilities, *Otto Rauh, Eberhard Stickel* 43

Modelling Multi-Party Activity in the IS Process from a Language/Action Perspective, *Ian R. McChesney* 53

Information System Analysis: A Formal Model for the Specification of Behavioural and Structural Class Properties, *Giovanni Rumolo, Maurizio Lenzerini* 63

On the Behaviour Modelling Using a Functional Approach, *Jaroslav Pokorny* 73

A Framework for Conceptual Exchange Between Object-Role Modelling and Extended Entity-Relatioship Modelling, *Phillip M. Steele, Arkady B. Zaslavsky* 83

Using an Explicit System of Concepts for Information Systems Modelling: COMOD, *Joao A. Carvalho, Luis Amaral* 93

Conjoined Co-evolutive Incrementalism for Information Systems Development, *Nagib Callaos, Belkis Callaos* 104

O/A-Matrix and a Technique for Methodology Engineering, *Kimmo Kinnunen, Mauri Leppanen* 113

Metamodelling: Concepts, Benefits and Pitfalls, *Mauri Leppanen* 126

SYSTEM DEVELOPMENT

Linguistically Based Information System Development - A Constructive Approach, *Erich Ortner* 141

Data Handling: A Perspective of Sign, Information and Meaning, *Jun-Kang Feng* 153

A Revised Spiral Model for Object-Oriented Development, *Egbert van der Walt, Annette L. Steenkamp* 163

An Approach for Information Systems Development, *Talib Damij* 173

Soft Systems Methodology (SSM): A Possible Way Forward? *Alan Hogarth, John Biggam* 183

The Application of Systemic Risk Analysis Techniques to Project Management Methodologies, *Roger W. Stewart* 192

Object Oriented Development Paradigm and its Deliverables, *Wita Wojtkowski, W. Gregory Wojtkowski* 200

A Language and Tool for the Engineering of Situational Methods for Information Systems Development, *Frank Harmsen, Sjaak Brinkkemper, Han Oei* 296

A Situation View of Software Development, *Maya Daneva* 215

Data Prototyping is Development, *Andrej Kovačič* 224

Problems in Introduction of the Rapid Application Development (RAD) Principles Put to Praxis, *Anton Ljubič, Stane Štefančič* 233

Towards Contingent Information Systems Development Approaches, *C. van Slooten, B. Schoonhoven* 242

BOOM - A First Step to an Object-Oriented Fourth Generation System, *Hubert Rumerstorfer, Josef Altmann* 254

Integration of Structured Methods and Formal Notations, *Pat Allen, Lesley Semmens* 264

QUELLE: A Knowledge - Based Approach to Systems Development, *Vlad Wietrzyk* 274

ORGANIZATIONAL ASPECTS

Key Issues in Information Systems Management, the Case of Poland, *Stanislaw Wrycza, Tomasz Plata-Przechlewski* 289

Some Recent Organisational and Social Changes Impacting upon Extant Information System Methodologies, *Stephen Keith Probert* 297

Impact of Solutions Implemented in Organization System on Information System, *Miro Jeraj* 307

STRATEGY PLANNING

Main Functions of Strategic Information System, *Sonja Treven* 315

A Process Model for Information Systems Strategy Planning, *Gregory Mentzas* 323

Strategic Aspects of Outsourcing, *Jochen Schwarze* 333

APPLICATIONS

IDA: an Intelligent Information System for Heterogeneous Data Exploration, *Maria Luisa Damiani* 345

The Development of a Database (FDB) for Information Exchange Among Clinical Engineers, *Aikaterini Krotopoulou, Georgia Panagopoulou, Spiros Sirmakessis, Paul Spirakis, Vassilis Tampakas, Dimitra Terpou, Athanasios Tsakalidis, Nicholas Pallikarakis* 357

The Development of an Information System for Human Resource Management, *Janez Jereb* 366

Strategic Information Systems: a New Perspective, *James Howell* 374

REENGINEERING

Accessing the Riches of the Past, *Francois J. Meijer* 389

G.O.A.T: Generic Object-Oriented Analysis Tool for Intelligent Object Extraction, *Gillian Sleith, Don McFall, John Hughes* 398

Organising the Redesign Process in System Development, *Kristin Braa, Tone Bratteteig, Leikny Ogrim* 407

Object and Program Reusability Mechanisms, *Xavier Castellani, Hong Jiang* 418

CASE TOOLS

Integration Issues of Information Engineering Based I-CASE Tools, *Karl, Kurbel, Thomas Schnieder* 431

An Empirical Study of Key Factors in CASE Implementation, *Erna Rupnik-Miklič, Jože Zupančič* 442

A CASE Tool for Re-engineering Conditional Logic, *John Bruce Cowan* 452

SPECIFIC ASPECTS OF IE

The Information Network System, *Ruud van der Pol, R.A.U. Quast* 463

Security Assessment During Information Systems Development, *Alenka Hudoklin, Branislav Šmitek* 471

Information as a Factor of Production: A Question of Systems Definition, *Bernd Schiemenz* 478

QUALITY

Experiences from an Information Resources Quality Assurance Program that Worked Well, but Ultimately Failed, *William A. Ohle, Margaret Ann Bartlett* 491

Measurement of Sex Dependent Quality Characteristics, *Vlado Vemuti, Peter Kokol, Ivan Rozman* 501

Software Quality Management as a Basis for Course Design, *Alastair Monger, Vladimir Petruv, Margaret Ross, Geoff Staples, Ian Tromans* 511

REQUIREMENTS ENGINEERING/ SYSTEM SPECIFICATION

HyperObjects: An Object-Oriented, 4th-Generation System Prototype, *Alois Stritzinger* 521

The Development of a Generic Requirement Specification, *Kim Trans* 528

An Object-Oriented Technique for Systems Specification, *J. Torres, J. A. Troyano, M. Toro* *538*

EDUCATION

An Environment for Research and Education in Information Systems, *Rommert J. Casimir* 551

SA++: How to Bring Organizational Aspects into Teaching Analysis and Design, *Jens Kaasboll* 559

A CASE Environment for the Teaching of Information System Principles, *Gary Allen, Adrian R. Jackson* 569

A Comparative Study of Internet Training Programs in the United States of America, *Lester J. Pourciau* 576

A VSM Based Plan for First Year Information Systems Course at the University of Paisley, *Angus Quin* 583

SPECIAL ASPECTS (COMPARISONS)

Cooperative Responding to Indirect Information Requests in Database Retrieval, *Xu Wu, Nick Cercone* 595

An Experimental Comparison of Object-Orientation and Functional-Decomposition for Communicating System Functionality to Users, *Tony Moynihan* 605

Complexity Reduction and Modelling Transparency in CASE Tools, *Sjaak Brinkkemper, Rolf Engmann, Frank Harmsen, Rob van de Weg* 615

The Use of Object Models for Information System Analysis, *Ying Liang, Mike Newton, Hugh Robinson* 625

Integrating Expert Systems Applications into Mainstream Business Operations, *Andrew Shie, Robert Moreton* 635

COMPUTER SUPPORTED COOPERATIVE WORK (CSCW)

JadeBird/III: A Collaborative Multimedia CASE Environment, *Fu-Qing Yang, Wei-Zhong Shao, Wei Li* 647

Semantic Model of Asynchronous Distributed Cooperation, *Igor T. Hawryszkiewycz, Leszek A. Maciaszek, J.R. Getta* 657

Cooperative Working for End-Users with MMTCA, *Sandy Kydd, Malcolm K. Crowe* 667

SOME TECHNICAL ASPECTS OF ISD - short papers

Evaluation of the Response Time of an Information System with Different Query Classes, *Aleksander Zgrzywa* 677

The Construction of a Personal Computer System Without a Command-Line Interface, *M.W.D. Maurer* 681

Computer Simulation Model for Cost-Benefit Analysis, *Miro Gradišar* 685

Database Structure and Disk Space Requirements of Computer Aided Management Systems, *Marek Milosz, Elizabeta Milosz* 689

A Comparative Analysis of Six Manufacturing Simulators, *Vlatka Hlupic* 693

WORK IN PROGRESS - short papers

Intelligent and Cooperative CASE Tool Kit for Development of Information Systems, *Maldonado A. Aguayo, Plaza A. Guevara* 699

The Purpose of Information System in the Computer Integrated Production Systems, *Anton Čižman* 703

Seeking the Actual Reasons for the "New Paradigm" in the Area of IS Analysis, *Vaclav Repa* 707

Humanization of Organization and Information Systems, *Duško Uršič* 711

Socio-Technical Systems Design Re-visited, *Christopher C. Wills* 715

Information Systems for Librarians and Users of Scientific Libraries, *Anna Maria Campanile* 720

A Conception of Project Information Space of the ES-GDSS Type, *Andrzej Zaliwski, Marian Kuraš* 723

Information Resources Integration Problems, *Adam Nowicki, Cezary Stepniak* 727

An Integrated, Fully Object-Oriented Software Development Method, *Benet Campderrich* 731

Using Experts Systems in Capital Investment, *Petr Wolf, Jana Drastikova* 735

Three-Level Concept in Data Modelling, *Jindrich Kaluža, Ludmila Kalužova* 741

APPENDIX

The Evolution of the Information Systems Field: The Impact of Technology on the Teaching and Practice of Information Systems Development, *A. Milton Jenkins (Invited Paper)* 747

AUTHOR INDEX

Aguayo, Maldonado A. 699
Allen, Gary 569
Allen, Pat 264
Altmann, Josef 254
Amaral, Luis 93
Bartlett, Margaret Ann 491
Biggam, John 183
Braa, Kristin 407
Bratteteig, Tone 407
Brinkkemper, Sjaak 206, 615
Callaos, Belkis 104
Callaos, Nagib 104
Campanile, Anna Maria 720
Campderrich, Benet 731
Carvalho, Joao A. 93
Casimir, Rommert J. 551
Castellani, Xavier 418
Cercone, Nick 595
Cowan, John Bruce 452
Crowe, Malcolm K. 667
Čižman, Anton 703
Damiani, Maria Luisa 345
Damij, Talib 173
Daneva, Maya 215
Doukidis, Georgios 3
Drastikova, Jana 735
Engmann, Rolf 615
Feng, Jun-Kang 153
Galliers, Robert D. 3
Getta, J.R. 657
Gradišar, Miro 685
Gričar, Jože 19
Guevara, Plaza A. 699
Harmsen, Frank 206, 615
Hawryszkiewycz, I.T. 657
Hlupic, Vlatka 693
Hogarth, Alan 183
Howell, James 374
Hudoklin, Alenka 471
Hughes, John 398
Jackson, Adrian R. 569
Jenkins, A. Milton 747
Jeraj, Miro 307
Jereb, Janez 366
Jiang, Hong 418
Kaasboll, Jens 559
Kaluža, Jindrich 741
Kalužova, Ludmila 741
Kinnunen, Kimmo 113
Kokol, Peter 501
Kovačič, Andrej 233
Krotopoulou, Aikaterini 357
Kuraš, Marian 723
Kurbel, Karl 431
Kydd, Sandy 667
Lenzerini, Maurizio 63
Leppanen, Mauri 113, 126
Li, Wei 647
Liang, Ying 625
Ljubič, Anton 233
Lybereas, Panagiotis 3
Maciaszek, Leszek A. 657
Maurer, M.W.D. 681
McChesney, Ian R. 53
McFall, Don 398
Meijer, Francois J. 389
Mentzas, Gregory 323
Milosz, Elizabeta 689
Milosz, Marek 689
Monger, Alastair 511
Moreton, Robert 635
Motschnig-Pitrik, Renate 31
Moynihan, Tony 605
Newton, Mike 625
Nowicki, Adam 727
Oei, Han 206
Ogrim, Leikny 407
Ohle, William A. 491
Ortner, Erich 141
Pallikarakis, Nicholas 357
Panagopoulou, Georgia 357
Petruv, Vladimir 511

Plata-Przechlewski, Tomasz 289
Pokorny, Jaroslav 73
Pourciau, Lester J. 576
Probert, Stephen Keith 297
Quast, R.A.U. 463
Quin, Angus 583
Rauh, Otto 43
Repa, Vaclav 707
Robinson, Hugh 625
Ross, Margaret 511
Rozman, Ivan 501
Rumerstorfer, Hubert 254
Rumolo, Giovanni 63
Rupnik-Miklič, Erna 442
Schiemenz, Bernd 478
Schnieder, Thomas 431
Schoonhoven, B. 242
Schwarze, Jochen 333
Semmens, Lesley 264
Shao, Wei-Zhong 647
Shie, Andrew 635
Sirmakessis, Spiros 357
Sleith, Gillian 398
Spirakis, Paul 357
Staples, Geoff 511
Steele, Phillip M. 83
Steenkamp, Annette L. 163
Stepniak, Cezary 727
Stewart, Roger W. 192
Stickel, Eberhard 43
Stritzinger, Alois 521
Šmitek, Branislav 471
Štefančič, Stane 233
Tampakas, Vassilis 357
Terpou, Dimitra 357
Toro, M. 538
Torres, J. 538
Trans, Kim 528
Treven, Sonja 315
Tromans, Ian 511
Troyano, J. A. 538
Tsakalidis, Athanasios 357
Uršič, Duško 711
van de Weg, Rob 615
van der Pol, Ruud 463
van der Walt, Egbert 163
van Slooten, C. 242
Venuti, Vlado 501
Wietrzyk, Vlad 274
Wills, Christopher C. 715
Wojtkowski, W. Gregory 200
Wojtkowski, Wita 200
Wolf, Petr 735
Wrycza, Stanislaw 289
Wu, Xu 595
Yang, Fu-Qing 647
Zaliwski, Andrzej 723
Zaslavsky, Arkady B. 83
Zgrzywa, Aleksander 677
Zupančič, Jože 442

PREFACE

During the past few years, many new concepts and approaches emerged in the Information Systems Development (ISD) field. The various theories, methods and tools available to system developers also bring problems such as choosing the most effective approach for a specific task. The aim of this *International Conference on Information Systems Development - ISD* is to establish an international forum for the exchange of knowledge, experience and new ideas, and to share and stimulate new solutions. This conference provides a meeting place for IS researchers and practitioners from Eastern and Western Europe, as well as those from other parts of the world. The Conference gives an opportunity for participants to express ideas on the current state of the art in information systems development methods and tools, and to discuss and exchange views about new methods, tools and their applications. An objective of the conference is not only to share scientific knowledge and interests but to establish strong professional ties among the participants. *The Forth International Conference on Information System Development - ISD'94* continues the concepts of the first Polish-Scandinavian Seminar on Current Trends in Information System Development Methodologies, held in Gdansk in 1988, and the Second and the Third International Conference on Information Systems Developers Workbench (Gdansk 1990, 1992).

In response to the Call for Papers, 95 papers from 26 countries were submitted for the conference. Papers were refereed by the members of the International Programme Committee helped by reviewers from these listed on the preceding page. In the final programme, 3 invited papers, 65 submitted papers and 17 presentations of the Work in Progress were included.

The co-chairmen would like to express their thanks to members of the Programme Committee for their suggestions and help in the preparation of the conference's programme. Together with other reviewers they also provided to authors many useful recommendations that greatly helped to improve the quality and relevance of the papers; their contribution was essential. We also wish to acknowledge the support given by the members of the Organizing Committee, the Association of Organizers of Work (Društvo organizatorjev dela Slovenije) and by the University of Maribor, School for Organizational Science.

We thank the Ministry of Science and Technology of the Republic Slovenia for its financial support of the conference. We also want to thank the sponsors of the conference: Soros Foundation - Open Society Fund, Ljubljana, Iskra, tovarna sestavnih delov Kranj (Iskra - Component Factory Kranj) and Pivovarna Union (The Union Brewery), Ljubljana.

We hope all participants in the conference will have their expectations met, and may every participant have a pleasant and fruitful stay in Slovenia

Jože Zupančič and Stanislaw Wrycza,
Co-chairmen of the Programme Committee

INFORMATION SYSTEMS PLANNING IN SMALL BUSINESS

Georgios I. Doukidis,[1] Panagiotis Lybereas[2] and Robert D. Galliers[2]

[1]Department of Informatics, Athens University of Economics and Business, 76 Patission Street, 10434 Athens, Greece
[2]Warwick Business School, University of Warwick, Coventry CV4 7AL, United Kingdom

Abstract

In this paper the use of Information Systems in the context of small business is discussed showing current research focus and presenting some parts of research areas which have not yet been tackled. The surrounding environment that facilitates the creation of small entrepreneurial businesses is further examined, together with the IS planning practices in such an environment. We find that these practices do not follow the conventional theories for IS planning. The main argument is that their management style and organizational structure determine to a significant extent the process and nature of IS planning. This relationship is envisaged under a broader perspective where we consider the inter-dependence between organizational structure and culture. We are proposing a shift of focus for IS planning research by suggesting a model which combines the above concepts.

1. INTRODUCTION

Small Businesses (SBs) comprise the vast majority of businesses throughout Europe. Indeed, during the last twenty years there has been considerable growth in the number and prosperity of SBs throughout Western economies, including Europe (Sengenberger et al. 1991). This growth, which is in line with the teachings of management theorists such as Drucker (1989) and Porter (1990), has been seen in both the manufacturing and services sectors. Across Europe, SBs are employing an increasing proportion of the total working population and are becoming increasingly identified with new products and new production processes, thus contributing to exports, national wealth and competitiveness.

Information Technology (IT) has made considerable inroads into large organizations. The majority of such organizations now rely on IT for their day-to-day operations. This diffusion of technology has been credited with significant cost reductions, gains in productivity and organizational effectiveness plus, in some cases, a definite competitive advantage (Earl, 1989). While considerable successes have been achieved, there have also been a number of technical and commercial disasters. Angell and Smithson (1991) argue that IT needs to be viewed in terms of both opportunities and risks, where the risks may outweigh the promised opportunities.

Research in the IS and management fields has concentrated until recently on the examination of large and successful corporations. Furthermore, it is often the case that these corporations operate in well established economies, where formal structures and processes are common. As Hofstede (1980) states: "Systems for management were mainly designed in the USA or in western Europe and, in addition, they were developed in a particular socio-economic and historical setting characterised by: competitive capitalism, political stability, extensive resources, overall economic expansion and non-intervention by the government".

As a consequence all proposed models and methodologies are based on data collected from large, well-established and well-managed firms operating in the above described environments. Raymond (1985) concluded that "very few scientific studies have been made in the specific context of small business".

Recently a number of IS and/or management researchers (such as, Pascale and Athos 1981, Slevin and Covin 1990, Drucker 1985, Mintzberg 1991) appear in favour of organizations with a more flexible and adaptive structure than the one identified in large companies. In most research agendas, we come across a shift of focus from large well-established corporations towards small businesses behaving in an adaptive, ad hoc manner. In the management area these new flexible forms of organizations are being examined through the lens of some organizational behaviour concepts such as 'entrepreneurship' and 'organizational culture'. All this change in research is due to the realisation that these emergent types of organizations will play an important role in tomorrow's economic and business global environment.

On the other hand, in the IS field, concepts such as 'Small Business Computing' have become common in research papers. Furthermore, an increasing number of such researches is being conducted in developing countries where different socio-economic and business structures are common, in comparison with the ones of countries where most IS models and methodologies have been developed (such as USA and western Europe). A major theme for research is the examination of the applicability of such models in these different environments. Although, it is traditionally assumed that small businesses should use the same management techniques as their larger counterparts (Raymond 1985), many researches using 'real' data have proved the opposite. For example, Kelmar and Noy (1990) conclude that planning techniques used in larger business were not necessarily appropriate for the smaller businesses, a similar conclusion to that of Brytting (1991) who states that the transference of existing management and organizational theory to the study of small businesses is problematic.

In this paper, the organizational characteristics of small entrepreneurial firms are presented extensively, stressing the important role of the top manager in the operation of such firms. The use of Information Systems in the context of small businesses is further discussed showing current research focus and presenting some parts of research areas which have not been tackled. The applicability of two widely-accepted IS stages-of-growth models is then tested in 24 cases operating in the Greek business environment, which is identified as one where small entrepreneurial firms flourish. The applicability problems are discussed, and an alternative way of examining these small businesses is being presented. The paper concludes by providing some hints for the need of a shift of focus towards context related research in the specific areas of small entrepreneurial businesses.

2. ORGANIZATIONAL CHARACTERISTICS AND MANAGEMENT ISSUES OF SMALL ENTREPRENEURIAL FIRMS.

2.1 Organizational characteristics of SBs

Small businesses are not just simplified large organizations. They are quite distinctive. They are effected by unique size-related issues causing a different interaction and analysis with their environments (Kelmar and Noy 1990, Raymond 1985).

The work environment identified in small businesses resembles, in general terms, the entrepreneurial work environment as described by Kets de Vries (1980). The main characteristics are described in table 1.

Table 1. Main characteristics of the entrepreneurial work environment.

Leadership style	• Autocratic / Directive
Decision making	• Centralised, • Lack of delegation, • Impulsive, • Lack of conscious planning, bold-proactive moves, mixture of operating and strategic decision making
Time horizon	• Short
Organizational environment	• High uncertainty, • Lack of sharing information
Suprastructure	• Poorly defined, absence of formal organizational chart
Infrastructure	• Frequently poorly defined or poorly utilised control and information systems, • Absence of standard procedures and rules, • No formalised systems (use of subjective, personal criteria), • Poor integration of activities, • Poorly defined job descriptions and job responsibilities (high degree of ambiguity).

2.2 The role of the top manager.

The role of the top manager is stressed in almost every research paper. West (1975) states that, "almost without exception, the small company is grossly understaffed, often being a one-man operation." The manager is the sole most important source of information and initiator of decisions and actions (Laios 1988). He/She is often perceived as operating under a different set of planning conditions than those which have been developed for the manager of the larger corporation (Kelmar and Noy 1990). Small businesses depend heavily on the abilities of a single person. As De Geus (1988) states: "the level of thinking that goes on in most management groups is considered below the individual manager's capacities. Autocratic institutions will learn faster or not at all - the ability of one or a few leaders being a risky institutional bet."

This means that the organizational culture of such enterprises provides the basis for a very flexible and effective organization. The dependence on one person however, puts success at risk, in contrast to collective processes where the chances of a wrong decision are deteriorated (in expense of other elements of course).

2.3 Small Businesses and the Planning process

According to Miller (1983) typical small businesses are generally run by an owner-manager and power is highly centralised. Strategy making in these firms tends to be rather more intuitive than analytical. It is performed by people who have a 'feel' for their business, not by staff planners and technocrats. Generally, time horizons are short and the focus is on operating matters rather than visionary master plans. The level of entrepreneurship is related to the personality, the power, and the store of knowledge of the leader. (Mintzberg and Quinn 1992).

Kelmar and Noy (1990) claimed that of those small business owners who actually undertake some strategic planning activities, some have been classified as proceeding in an unstructured, irregular, incomprehensible, incremental, and reactive fashion. Furthermore, the small business planning function has also been described as being ad hoc and informal; unsystematic and non-explicit; fragmented and heuristic. Some of the reasons for this include: little training, lack of necessary skills, lack of time and resources, and a fear of planning itself. Certain owners consider planning irrelevant, unnecessary and too time consuming.

As already stated, the top manager plays a uniquely important part in the running of smaller business, and his/her personal influence has a greater impact than his/her counterpart in the large firm (Martin 1989). Small firm owners / manager do not engage in systematic planning. Planning in the small firm has the following characteristics:

- often done on an ad hoc, problem basis,
- frequently only a mental activity of the owner / manager,
- informal, sporadic, and closed,
- often relying on advice from random acquaintances with less skill and / or less experience than the owner himself. (Robinson 1982)

3. INFORMATION SYSTEMS IN THE CONTEXT OF SMALL BUSINESS.

Information Systems have become recently the subject of examination of researchers in the context of small businesses, due to their importance for these companies throughout all their stages of existence (from early survival struggles to maintaining a competitive edge).

3.1 Differences between Small Business and large organizations.

Raymond and Magnenat-Thalmann (1987), comparing small and large firms, conclude that: "Small businesses face many of the same problems and decisions as large firms but without the benefit of staff expertise and multiple managerial levels. Thus the top manager or managers must shoulder a much broader decision making burden both horizontally and vertically. As a consequent, information to support managerial decisions should be as important, if not more, to the small business manager as to his large firm counterpart." Forest and DeCarlo (1984) suggest that, the small business that can manage its information needs has the strategic edge. They conclude that, the growing business needs to solve the problem of getting the right data to the right person at the right time, and in the right form.

Concerning the use of IT, Heikkila, Saarinen and Saaksjarvi (1991) propose three major differences between SBs and large organizations:

- SBs tend to use computers more as tools and less as a communications medium;
- the few stakeholders involved in SBs mean that there are likely to be fewer problems in terms of organizational politics; and
- SBs have much fewer resources available to implement IT solutions.

While it is hard to dispute the lack of resources, the other points are less clear-cut. Firstly, the treatment of computers as tools may only be a temporary phase as networks promise considerable gains for SBs in terms of collaboration with other firms. Lacking their own resources but with normally a plentiful supply of entrepreneurialism, SBs rely more on short-term sub-contracting and other temporary arrangements to satisfy the needs of their customers. Secondly, SBs are not totally free of organizational conflict and family politics can be just as bitter as the politics found in bureaucracies.

3.2 Potential advantages possessed by small business.

Wroe (1987) argues that small firms possess certain potential advantages in making use of the technology, since they are able to complete the transition process much faster and they possess greater flexibility to undertake any re-organization required to realize the full benefits of the technology (Poutsma and Walravens, 1989). Furthermore, the flexibility of new technologies facilitates small batch, tailored or niche-focused production, the preserve of the small firm. Through accurate and systematic record-keeping, IT can help SBs in areas of traditional concern: the collection of outstanding payments, stock control, increased sales, and improved after-sales service. Although Lincoln and Warberg (1987) found that, in practice, SBs failed to utilize the marketing data that they possessed, Poutsma and Walravens (1989) argue that IT can help an SB to develop its markets, increase turnover, raise profitability, and still remain a small firm, able to realize the benefits of the smallness in service and flexibility.

These views are further supported by Cornford and Whitley (1991) who identify the following benefits IT offers to SBs:

- improved productivity and performance within the enterprise;
- greater internal control of operations;
- the possibility of new ways of managing activities;
- improved management perception and penetration of business environments;
- the possibility of new organizational forms (e.g. networked organizations);
- the delivery of a valued and high quality package of product and service;
- the extension of available markets;
- the redefinition of existing businesses or the spawning of whole new businesses.

3.3 Problems in adopting and using IT.

On the other hand, SBs have particular problems in adopting and using IT. Typically they do not have the appropriate skills available in-house and thus have to train existing staff or purchase those skills in the marketplace. IT is usually associated with a systematic approach to management and decision-making and its introduction requires careful planning, whereas much SB management practice is based on short-term, informal, ad hoc lines (Hill et al. 1984). Although the technology is much cheaper than before, it still represents a considerable investment for SBs, who traditionally lack such funds. The introduction of IT, which may lead to dramatic changes in the business's fundamental activities, requires an awareness and basic knowledge within the management function but many owners of SBs appear to be too busy 'surviving' to invest time in such projects. Therefore, there is a significant risk that such efforts to introduce IT will be unsuccessful and the cost of such a failure may be fatal for the small firm lacking adequate financial and productive cushioning (Scholhammer and Kuriloff 1979). Thus, it is hardly surprising that until recently many SBs have avoided such risks by ignoring IT.

3.4 Top management involvement and organizational culture.

Despite the reported importance of IS in the context of small businesses, the top managers / owners do not seem to realize the potential benefits of IS. A large number of studies (e.g. Alpar and Ein-Dor 1991, Doukidis et al. 1994a) indicate that IS is regarded just for operational and tactical activities and it is not viewed as a strategic weapon.

A serious and much discussed issue for IS is the difficulty to secure top management commitment and involvement (Lederer and Sethi 1992; Martin (1989) identified a continuum of five behavioural patterns of top management involvement in the IS function. At one end of this continuum the top manager is completely remote even from key decisions, while at the other end he/she routinely interacts directly, hands-on, with the IS function. Cragg and King (1992) applied Martin's typology and concluded that the more closely top managers are involved in computerisation, the more IT growth is experienced.

A 'high tech' organizational culture, supported by a committed and involved top manager, and a vision for IS exploitation (Sambamurthi et al 1993) prove to be key elements for IS growth in such environments. As Salmela and Ruohonen (1992) argue, the flexibility of such companies renders them good candidates for IS development since they have the potential to adapt to the required changes. However, the same culture can become an impediment to prospective changes. In other words, success in this specific structure with the owner leading all decision making is dependent on the capabilities of this single person. It is evident therefore that there is a thin line between success and failure with respect to IS in the context of small businesses.

19

4. EXAMINING IS GROWTH IN SMALL ENTREPRENEURIAL FIRMS.

4.1 The need.

As already outlined in the previous section, most of the published researches on IS in small businesses concentrate on the identification of success factors and inhibitors of IS implementation (for example Doukidis et al 1994a, Verber and Zupancic 1993, Yap et al 1992).

At the same time numerous researches have outlined the role of the top manager in every aspect of the IS function while realising that this person acts in an ad hoc manner. Despite the importance these studies place on this role, the decision making process has not been explicitly researched. There exist some references on the nature of IS planning in the context of small business, but no real in-depth analysis of this issue can be claimed to have taken place.

In the ongoing effort to provide an integrative framework for the management of the IS function, various IS stages-of-growth (SOG) models have been developed, describing a series of stages that a company's IS planning activities are supposed to go through. A company can be positioned in such a model by observing its current planning activities, while the model itself provides a 'desirable' future state. The applicability of these models is questioned in the context of small entrepreneurial businesses. Since these behave in an ad hoc fashion it is expected that these models will prove inadequate for these cases.

The most well known IS SOG model has been developed by Nolan (1979, Gibson and Nolan 1974). This model has been widely used in consulting and initiated a debate in the research community. It is based on the evolutionary hypothesis that businesses and business functions, in this case the IS function, follow a more or less common pattern of growth. Nolan mapped this growth upon six stages and described it with four factors (i.e. applications portfolio, DP organization, DP planning and control, and user awareness). The principal argument is that the evolution of the IS function is mirrored on the rate of the respective expenditure which follows an S-shaped curve over time. The framework is detailed and provides even benchmarks for its application.

However, Nolan's model has been widely criticized for both its theoretical assumptions and its empirical validity. In addition, the emphasis on technologies of the 1970s dates the model. Since then several efforts have been made to formulate alternative IS SOG models focusing on specific factors for describing the evolution. If the model is to address the IS function as a whole, the set of factors has to deal with every aspect of it and yet be concise and easily applicable. The IS SOG model presented by Sutherland and Galliers (1989, Galliers and Sutherland 1991) (hereafter referred to as Galliers' model) attacks this issue by drawing on the 7S framework (Pascale and Athos 1981) which was initially proposed for understanding and managing organizations in a holistic manner. This 'managerial molecule' comprises of the following seven elements: strategy, structure, systems, skills, staff, style, superordinate goals. Galliers defined these elements in terms of the IS function and formulated a six stage model for describing and prescribing managerial action for growth.

As part of a larger research effort (Lybereas et al. 1994, Doukidis et al. 1994b), this paper examines small businesses and the applicability of Nolan's and Galliers' IS SOG models. Despite the considerable criticism on stages-of-growth models, they are widely used internationally. It is therefore important to examine their applicability in the specific context of small entrepreneurial businesses.

4.2 Research Methodology.

In particular, 24 typical small businesses in Greece were examined in great depth. All the small businesses examined in our research started out as small family businesses. Today they have grown significantly but they still retain many characteristics of the family business mode of operating. In fact, many of them are still family businesses despite their being among the leaders in their sectors.

During a period of one year these companies were visited at least three times each. Separate open-ended questionnaires were prepared for the application of each model. The questions asked emphasized on the present situation, on the evolution in the past, as well as on future plans. Since these models map the growth of the IS function over time, it would not be enough to identify only the present stage of development. Although historical information can be collected more effectively in longitudinal case studies, we paid particular attention to this issue. We were able to spent sufficient time with more than one person in each company and therefore get a quite clear picture of the evolution of the IS function.

The primary reason for selecting Greece as the country for examination is that it combines many characteristics facilitating the existence of small entrepreneurial firms. In Greece large well-established, and well-managed (according to international theories) firms are extremely rare. The size of companies is quite small - something like 99% of industrial companies employee less than 100 people. The importance of small businesses for the Greek economy is therefore extremely relevant.

There exist fundamental differences in the management style identified in Greek companies when compared with other western European or American companies. Various cross-country researches clustering countries with respect to managerial characteristics categorise Greece in an unexpected way, that is not according to historical, geographical or linguistic factors. Hofstede (1980) has published the most important and widely accepted work on cultural differences between 53 countries. His measures, denoting the differences between cultures, show extensive differences between Greece and USA or other Western European countries, stressing its unique cultural environment. Several other researches have reached similar conclusions (Triandis and Vassiliou 1972, El-Namaki 1990, Bourantas et al. 1990, Veiga and Yanouzas 1991). These results suggest the need for context related research in the IS field as has been also explicitly advocated by others (e.g. Avgerou and Doukidis 1993).

From the 24 case studies we identified two major trends for their IS evolution:

A. Companies which succeeded in responding to the challenges of IS, while their growth with respect to IS can be characterised as unpredictable; and
B. Companies that did not prove flexible and adaptable enough to adopt IS successfully and remained in the early stages of the two models.

4.3 Case A: Unpredictable growth with respect to IS

Some of the successful companies with respect to IS have introduced IT as early as the '60s or '70s. The development of new systems has not been as planned as it has been opportunistic. Most of these businesses have been successfully exploiting advanced technologies without having surpassed the problems outlined in the initial stages of the two models. Today their systems are quite advanced and in some cases very sophisticated. For example, in one of the leading dairy companies, there exist several innovative systems such as computer-coordinated fully-automated inventory control, hand held computers and electronic links to their advertising services supplier. At the same time, their basic applications infrastructure is not at all sophisticated or high level. Companies such as this have proceeded in an unsystematic, opportunistic manner.

The organization of the IS function has also been flexible with direct involvement of the top manager/owner. With respect to Nolan's and Galliers' models it is very difficult to match this evolution with the 'normal' course that is described in them. For instance, the very introduction of IT (stage 1) has been a matter of CEO/owner intervention in an entrepreneurial effort to improve operations and competitive position. Such characteristics are given only at stage 5. The operations and systems are flexible and informal, which in the context of the two models can be mapped either as a characteristic of the early stages or as evidence of a mature flexible organization. Because the later implies experience and sophistication in both IS and general management, we have to position these companies at stages 2 and 3. This in turn is not satisfactory for several reasons (e.g. the structures and processes found, although seemingly primitive accomplish more than the models allow in the respective stages).

The staff employed has been growing as the infusion of IT expanded and consisted mainly of programmers/analysts in one role. The IS manager in most cases has always been both a technical person and an IS decision maker. In addition, the CEO/owner has always had hands-on involvement in the decisions about IS in cooperation with the IS manager. This means that without having the specialization of skills and staff described in the models, these companies performed essentially the same tasks even in an informal and lower level manner.

It is important to note that in general, such companies display characteristics of almost all stages in both models. In other words, it is difficult to point at one or two stages as reasonably representative of any of these companies without being unfair.

4.4 Case B: Unadventurous companies with respect to IS

With respect to the general business characteristics, these companies are similar to the ones described above: they are family-owned entrepreneurial firms. The difference lies in the way they have proceeded with IS. There are of course examples where computerisation failed or never really took off. So, for example, companies whose efforts have been just as ad hoc and opportunistic as mentioned above, they were not able to succeed and to yield any real or lasting benefits. This is why they give the impression of failures in what concerns the adoption and dissemination of IT. This is definitely not due to its recent introduction. Computerisation in these companies has a relatively long history but a stagnant one. In few cases some growth has been experienced in the past but it was not sustained. For example, a well established cosmetics company introduced Data Processing in the 1960s and until the mid '70s covered most operational and financial business areas that were common to be automated at that time. Since then, virtually no development has taken place and they are even reluctant to upgrade their hardware finding their supplier very pressing.

In all of these cases nobody is currently worried about improving and expanding any benefits from IS. As a result, they are stuck in the first two stages of the models for a long time now. The 'strange' fact is that they are all well established and growing companies in their sectors. This is not exceptional however, since many researchers (e.g. Strassmann 1985, Cragg and King 1992) have found no evidence to support a positive relation between IS sophistication and firm performance. This ineffectiveness in deploying IS could possibly be attributed to the relative unawareness from the part of the owner of the key issues related to IS and IS management. This is also true for the rest of the members of the organization and even if someone has enough knowledge, he/she is not in position to initiate any change.

Summarising, what has happened in these particular companies is that their owners have been quite effective in managing the business domain but they lacked the knowledge and missed the opportunity to exploit IS equally well.

5. CULTURE'S CONSEQUENCES ON IS PLANNING

5.1 IS stages-of-growth models revisited.

Apart from the problems presented above with respect to the applicability of the two IS stages of growth models, we deliberately left out of the previous comments the soft elements of Galliers' model (i.e. skills, style and superordinate goals) because they deserve special discussion. The first point that has to be made clear is that by their nature they are difficult to operationalize and diagnose. This can first be seen on the generic descriptions given in the model. Moreover, the situations found in practice were difficult to match with any of these descriptions. All the companies logically have to be positioned to the early stages but the phenomenon of identifying characteristics which span across all stages is more apparent here than in any other element. Since high-level decisions are more or less informal, unsystematic and unplanned it becomes clear why the history of these companies does not follow the descriptions of Nolan's and Galliers' models. The fact that these models virtually mirror how a 'good' western corporation is organized, explains why the present situation of the *opportunistic* and *entrepreneurial* small businesses cannot be matched consistently with any of the stages.

BEST COPY AVAILABLE

It is an interesting coincidence that Sutherland and Galliers (1989) have used the same two terms to describe the Superordinate Goals of stage 5. This is in accordance with other researchers (e.g. Isenberg 1987, Ciborra 1991) who suggest that large organizations should relax their rigidities and act in a way similar to smaller enterprises in order to compete more effectively. Anyway, it emphasises the point that the model is appropriate almost only for firms of a certain type. The small companies on which we focus are inherently opportunistic and entrepreneurial whether at an early or later stage. Moreover, as they grow and become more mature there are chances that they transform to more formal and bureaucratic.

5.2 Linking organizational culture and structure in IS

The entrepreneurial and opportunistic attitude towards IS of Group A companies seems to explain to a significant extend the unique growth patterns of these companies. Indeed several researchers have shown that management style is a significant determinant of firm growth and success. However, entrepreneurial attitude is not desirable and effective in its own right. Slevin and Covin (1990) suggest that any type of management style must be coupled with an appropriate organizational structure. They distinguish between entrepreneurial and conservative management style and between organic and mechanistic structures. According to the way they have defined these terms they concluded that entrepreneurial firm behaviour correlates positively with firm performance in the presence of an organic organizational structure and negatively in the presence of a mechanistic one. Although the idea of combining the two concepts is very insightful, there is a general lack of consensus around the definition of these terms.

We applied their model positioning each Group according to the management style and organizational structure of the IS function, adapting the measures of style and structure to the IS field. In particular, we classified a specific Group as having an entrepreneurial management style with respect to IS, when its management's behaviour is characterised by risk taking, proactivity and innovation. A conservative management style towards IS is, on the other hand, reactive or even myopic. Organic structures with respect to IS are characterised by open communication, adaptation, consensus and loose control, while mechanistic ones tend to be more traditional, tightly controlled and hierarchical (Slevin and Covin, 1990). We applied their model positioning each Group according to the management style and organizational structure of the IS function, as shown in Figure 1.

ORGANIZATIONAL STRUCTURE

MANAGEMENT STYLE	MECHANISTIC	ORGANIC
ENTREPRENEURIAL	Pseudo-Entrepreneurial Firms (2) Group A	Effective Entrepreneurial Firms (1) Group A
CONSERVATIVE	Efficient Bureaucratic Firms (3)	Unstructured, Unadventurous Firms (4) Group B

Figure 1: Slevin and Covin's model

From the figure above we can see that none of our cases was an efficient bureaucratic firm (cell 3) since these are usually large organizations. As far as the organizational structure is concerned, no mechanistic Group B companies were identified, while there was only a limited number of mechanistic Group A firms. This grid provides a very representative characterisation of the observed behaviour of the companies in each group. It suggests that not all entrepreneurial firms are expected to be successful and neither will all conservative companies be inefficient. Management style alone cannot be a determinant of business behaviour and effectiveness. The coupling of managerial style with organizational structure justifies different behaviours displayed by equally entrepreneurial firms in Greece. In particular, Group B and some of Group A companies are not successful with the use and management of IS because they failed to match an appropriate organizational structure with the given managerial style, or vice versa.

5.3 Explaining IS growth with mobility on the Style/Structure grid.

The most important feature of the model adapted is that it incorporates the notion of 'mobility'. According to Slevin and Covin successful organizations manage to cycle between Cells (1) and (3), by achieving simultaneous changes of both their management style and organizational structure. On the other hand, unsuccessful organizations get stuck in the transition states of cells 2 or 4, a situation caused by 'dysfunctional' cycling occurring when only one dimension is changing.

Similar transitions are implicitly embedded in the IS Stages-Of-Growth models. The course of evolution for these models follows a path with an initial slack in formality and control, moves on to a period of rationalisation and then proceeds to a planned flexibility. In principle, SOG models prescribe an unidirectional evolution and although backward moves are theoretically acceptable, they are not accommodated in the models. Slevin and Covin's grid on the other hand, allows us to study in more depth and with higher flexibility the possible moves that firms can make along the dimensions of style and structure. IS SOG models and the Style / Structure grid complement each other and their combined examination leads to an enriched picture of the IS profile of a company.

Placing the general course of evolution as prescribed by the stages-of-growth models on to the Slevin and Covin framework, we can observe a transition from cell 4 (SOG stages 1 and 2: an unstructured, unadventurous state of the organization) to cell 3 (SOG stages 3 and 4: tightly / formally controlled environment) to cell 1 (SOG stages 5 and 6: a more adaptive, entrepreneurial organization). As already mentioned, Slevin and Covin describe such a movement as one of the successful transition possibilities. In addition they expand this unidirectional concept of the SOG models to suggest other possible patterns.

In our cases we have identified two specific movements between the four cells of Slevin and Covin's model:

- Between cells (1) and (4). *Walking on thin ice.*

There exist various organizations mainly from Group A (cell (1)) that are changing their entrepreneurial management style with respect to IS and are becoming more conservative. These organizations are mainly the ones that achieved an 'unexpected' IS success and can not foresee any further advantages obtained from extending their IS activities. If they stay idle for too long they will become unstructured, unadventurous firms, and will slide to cell (4). In reverse, there are some Group B companies which changed their management style when a new employee , usually a member of the family who owns the company, became involved in the company's management and his/her innovative ideas on IS implementation transformed the company to an effective entrepreneurial firm (cell 1). In general, therefore, this movement concerns organic firms balancing success and failure with respect to IS. The management style changes between entrepreneurial and conservative either when there is a change of management implementing new ideas, or when external factors influence the conduct of the company.

- Between cells (1) and (2). *Continuous opportunity seeking.*

This movement describes a constantly entrepreneurial management style while the organizational structure may change. Typical examples include companies which started off in an extremely entrepreneurial and organic fashion. Due to unexpected and rapid expansion, they became quite mechanistic and not adaptive to changes in the environment, by imposing tight formal controls on its operations. At the same time, their IS management remained entrepreneurial, but was restricted by formal bureaucratic procedures (cell 2). Of course, cell 2 is a transitory cell and the companies that find themselves with a mechanistic organizational structure and an entrepreneurial management style are bound to change in a short period of time. This may occur either because the entrepreneurial style wears out and the managers end up in conservative processes (move to cell 3) or the entrepreneurial style persists and forces a gradual transformation of the structure to a more organic one (move to cell 1).

The two possible types of movement identified above indicate the reasons for the problematic applicability of Nolan's and Galliers' 7S models in the context of SBs, since in practice the observed movements are not represented in these models. Both identified movements (between cells 1 and 2, and 1 and 4) are, according to Slevin and Covin, characteristics of dysfunctional cycling, where only one dimension changes. The larger conclusion that can be drawn from this empirical evidence is that the stages-of-growth models do not cover all possible movements, as these can be identified by the Slevin and Covin grid. Figure 2 shows the various movements identified.

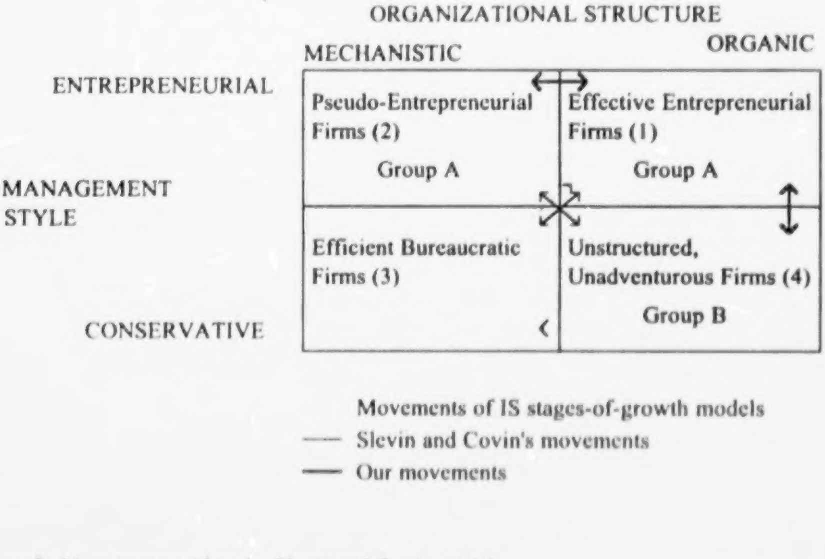

ORGANIZATIONAL STRUCTURE
MECHANISTIC
ORGANIC
ENTREPRENEURIAL
MANAGEMENT STYLE
CONSERVATIVE
Pseudo-Entrepreneurial Firms (2)
Group A
Effective Entrepreneurial Firms (1)
Group A
Efficient Bureaucratic Firms (3)
Unstructured, Unadventurous Firms (4)
Group B
Movements of IS stages-of-growth models
Slevin and Covin's movements
Our movements

Figure 2: Movements within the Slevin and Covin model

6. The New Model

6.1 Criticism on the Slevin and Covin's grid.

Even Slevin and Covin's model is quite restrictive. Two problematic areas can be identified:

- **Simplistic Representation.**

The distinction between entrepreneurial and conservative management style, as well as between mechanistic and organic structure is rather simplistic. Further analysis is called for in order to scrutinise the meaning and measurement of these two dimensions. In the context of our cases, although entrepreneurial and conservative management style towards IS can be quite easily distinguishable, since it depends on the ideas and actions of a single person, the classification of organizational structure with respect to IS was rather hard to make. Another issue concerns the complexity of organizations and the fact that different parts of the whole system may have various styles and structures, thus making it difficult to characterise the organization as a whole. This problem became apparent in several of our cases, where the simple distinction between mechanistic and organic organizational structure was inadequate.

- **Level of analysis.**

Despite the fact that Slevin and Covin's model provides insight into the 'peculiarities' observed for small businesses, the whole analysis still remains at a descriptional level. As is the case with the two SOG models, Slevin and Covin's grid simply presents from a different perspective the actual 'symptoms' observed in the two groups, giving some initial explanation about the placement and movement of the companies with respect to the SOG models (by stressing the importance of fit between organizational structure and management style). It is evident that Slevin and Covin's grid does not really examine the 'causes' underlying the 'symptoms', it merely presents the later in a different way.

6.2 Explaining the causes behind the symptoms.

In order to overcome the two limitations of Slevin and Covin's grid mentioned earlier, we propose the examination of a new model based on the basic principles of this grid, but focusing further on the causes behind the symptoms.

In particular, we can overcome the first issue of simplistic representation, by replacing the dimension of organizational structure (which is the most problematic) with Pascale and Athos's 7S model, which provides a holistic representation of the organization. The examination of the seven characteristics making up their 'managerial molecule' (i.e. strategy, structure, systems, staff, style, skills, and superordinate goals) provides a much broader understanding of the organization and its activities.

As far as the issue of the level of analysis is concerned an extension of the model to incorporate theory on management behaviour can be considered. The field of organizational behaviour has a long history and remarkable work has been done on the examination of issues focusing on the causes behind observed organizational symptoms. One such concept underlying all organizational actions is the issue of: 'organizational culture'. Hofstede's research work (1980, Hofstede et al. 1990, 1993) is the most widely accepted and cited work in this particular field. The major strength of his work is that the organization is not examined in isolation to its surrounding environment. In fact, the national level within which the organization operates, as well as, the individual level describing the employees' values and practices, are considered in detail. Therefore, Hofstede's work provides a useful tool for analysing the causes behind organizational actions by examining the later in relation to the national environment and the individual characteristics.

The new model which combines Hofstede's work and the 7S model, provides a refined extension to Slevin and Covin's framework. This model is ideal for examining the causes underlying the observed symptoms, which are better described by the use of the 7S model.

7. CONCLUSIONS AND FURTHER RESEARCH

Information Systems and Small Businesses is a common theme for current research. The recent focus on these issues is mainly due to the realisation of the importance that SBs play for the worldwide economy. At the same time, various IS models have been developed examining the IS function. The applicability of such models in Small Businesses has recently begun to be questioned. The focus of this work was to examine two widely accepted IS stages-of-growth models in the context of SBs.

The primary outcome of our case studies is that Nolan's and Galliers' IS SOG models are not universally applicable, but only in specific contexts of organizational structure and style, namely large formal organizations found in developed western economies. Although

Galliers' framework contains soft elements (style, skills, superordinate goals) which try to account for contextual attributes, their elaboration is limited to the management of the IS function and to describing managerial attitudes matching the rest of the elements. Ping and Grimshaw (1992) have applied the same models in China. It is important to notice that despite the fundamentally different culture of China in relation to Greece, they reach to similar conclusions regarding the significance of culture in IS growth. This surfaces the fact that any model or theory is susceptible to the context in which they are made. Both models have been developed and tested on organizations which are structured and function according to principles which are common to North American and Western European economies. To transfer the knowledge and experience encapsulated in these models to other companies is risky if the context is different.

In order to explain the peculiarities observed in the application of these two models we used Slevin and Covin's grid mapping our cases with respect to their organizational structure and management style towards IS. This placement assisted in explaining these symptoms by observing the level of fit between structure and style. This is simply another representation of our cases without doing an in-depth analysis of the causes underlying these symptoms.

The need for the development of a new model considering the causes underlying the observed symptoms is evident. Also, the specific needs and characteristics of small businesses should also be incorporated in such a model. Early results reported in other two research papers (Lybereas et al. 1994, Doukidis et al. 1994b) include considerations on these issues, while the new research model mentioned above and currently being under development could prove to be quite important in the understanding of small business and the way they operate.

Acknowledgments

The authors would like to thank Mr. Nikos Mylonopoulos for his contribution is this larger research effort.

8. REFERENCES

Alpar, P., and Ein-Dor, P., (1991), "Major Information Systems Concerns of Entrepreneurial Organizations", Information and Management, Vol. 20, pp. 1-11

Angell, I.O., and Smithson, S., (1991), "Information Systems Management: Opportunities and Risks", Basingstoke, UK: Macmillan

Avgerou, C., and Doukidis, G.I., (1993), "Information systems in Greece: the need for context related research", European Journal of Information Systems, pp. 69-77.

Bourantas, D., Anagnostelis, J., Mantes, Y., and Kefalas, A.G., (1990), "Culture gap in Greek management", Organization Studies, pp. 261-283

Brytting, T., (1991), "Spontaneity and Systematic Planning in Small Firms - a Grounded Theory Approach", International Small Business Journal, Vol. 9, No 1, pp. 45-63

Ciborra, C.U., (1991), "From thinking to tinkering", Proceedings of the 12th International Conference on Information Systems, New York, December, pp. 283-291.

Cornford, A., and Whitley, E., (1991), "A Review of IT Strategies within the UK SME Sector", Internal Research for the ESPRIT project smEsprit (No.5639), The London School of Economics

Cragg, P.B., and King, M., (1992), "Information systems sophistication and financial performance of small engineering firms", European Journal of Information Systems

De Geus, A.P., (1988), "Planning as learning", Harvard Business Review, March-April, pp. 70-74.

Doukidis, G.I., Smithson, S., and Lybereas, T., (1994a), "Trends in Information Technology in Small Business", Forthcoming in the Journal of End User Computing., An early version of the paper was in the Proceedings of the 13th International Conference on Information Systems, Dallas, Texas, 13-16 December, pp. 139-148.

Doukidis, G.I., Mylonopoulos. N.I., and Lybereas, P., (1994b), "Information Systems Planning within Medium Environments: a critique of IS growth models", International Transactions of Operational Research, Vol., 1, No., 3, pp. 293-303

Drucker, P.F., (1985), "Innovation and Entrepreneurship", Pan Business Management.

Drucker, P.F., (1989), "New Realities", New York: Heinemann

Earl, M., (1989), "Management Strategies for Information Technology", New York: Prentice Hall

El-Namaki, M.S.S., (1990), "Small business - The myths and the reality", Long Range Planning, Vol. 23, No.4, pp.78-87

Forest, R.B., and DeCarlo, F.D., (1984), "Computer Information Strategies for Smaller Corporations", Inc., November, pp. 129-164

Galliers, R.D., (1991), "Strategic Information Systems Planning: Myths, Reality, and Guidelines for Successful Implementation", European Journal of Information Systems, Vol. 1, No 1, pp. 55-64

Galliers, R.D., and Sutherland, A.R., (1991), "Information systems management and strategy formulation: the 'Stages of Growth' model revisited", Journal of Information Systems, No. 1, pp. 89-114.

Gibson, C.F., and Nolan, R.L., (1974), "Managing the four stages of EDP growth", Harvard Business Review, January-February, pp. 76-88.

Harrison, W.L., and Farn, C-K., (1990), "A comparison of Information Management Issues in the USA and the Republic of China", Information and Management, No 18, pp. 177-188

Heikkila, J., Saarinen, T., and Saaksjarvi, M., (1991), "Success of Software Packages in Small Businesses: An Explaratory Study", European Journal of Information Systems, Vol. 1, No. 3, pp. 159-169

Hill, S., Blyton, P., Patrick, C., and Peregrine, J., (1984), "Decision making in the South Wales Engineering Industry: A Survey", Cardiff: UWIST Discussion Papers in Business and Economics, UWIST

Hofstede, G., (1980), "Culture's consequences: international differences in work related values", Beverly Hills, California: Sage.

Hofstede, G., Neuijen, B., Ohayv, D.D., and Sanders, G., (1990), "Measuring Organizational Cultures: A Qualitative and Quantitative Study across Twenty Cases", Administrative Science Quarterly, Vol. 35, June, pp. 286-316

Hofstede, G., Bond, M.H., and Luk, Ch.-L., (1993) "Individual Perceptions of Organizational Cultures: A Methodological Trestise on Levels of Analysis", Organization Studies, Vol. 14, No. 4, pp. 483-503

Isenberg, D.J., (1987), "The tactics of strategic opportunism", Harvard Business Review, March-April, pp. 92-97.

Kelmar, J.H., and Noy, S., (1990), "Perceptual differences in small business strategic planning", Fifth National Small Business Research Conference, University College of Southern Queensland, Toowoomba, June.

Kets De Vries, M.F.R., (1980), "Organizational Paradoxes: Clinical Approaches to Management", Tavistock Publications

Laios, L., (1988), "Emerging Information Technologies for the Small and Medium Enterprises", in "Information Technologies for Organizational Systems", Bullinger H.J., et al., (eds.), pp. 1129-1132

Lederer, A.L., and Sethi V., (1992), "Meeting the Challenges of Information Systems Planning", Long Range Planning, Vol. 25, No 2, pp. 69-80

Lincoln, D.J. and Warberg, W.B., (1987), "The role of microcomputers in small business marketing", Journal of Small Business Management, Vol. 25, No. 2, pp. 8-17

Lybereas, P., Mylonopoulos, N.A., Doukidis, G.I., and Galliers, R.D., (1994), "Information systems planning and entrepreneurial culture in medium-sized enterprises", In the Proceedings of the 2nd European Conference on Information Systems, (Ed. W., Baets), pp. 23-32

Martin, C.J., (1989), "Information Management in the Smaller Business: the Role of the Top Manager", International Journal of Information Management, No 9, pp. 187-197

Miller, D., (1983), "The Correlates of Entrepreneurship in Three Types of Firms", Management Science, Vol. 29, No 7, pp. 770-791

Mintzberg, H., (1991), "The Effective Organization: Forces and Forms", Sloan Management Review, No 54, Winter, pp. 54-67

Mintzberg, H., and Quinn, J.B., (1992), "The Strategy Process: Concepts and Contexts", Prentice-Hall International Editions.

Nolan, R.L., (1979), "Managing the crises in data processing", Harvard Business Review, March-April, pp. 115-126.

Pascale, R.T., and Athos, A.G., (1981), "The art of Japanese management", Penguin books.

Ping, Z., Grimshaw, D.J., (1992), "A comparative study of the application of IT in China and the West: Culture and the Stages of Growth model", International Journal of Information Management, pp. 287-293.

Porter, M.E., (1990), "The Competitive Advantage of Nations", Harvard Business Review, March/April, pp. 73-93

Poutsma, E., and Walravens, A., (eds.), (1989) "Technology and Small Enterprises: Technology Autonomy and Industrial Organization", Delft, Holland: Delft University Press

Raymond, L., (1985), "Organizational Characteristics and MIS Success in the Context of Small Business", MIS Quarterly, March, pp. 37-52

Raymond, L., (1989), "Management Information Systems: Problems and Opportunities", International Small Business Journal, Vol. 7, No 4, pp. 44-53

Raymond, L., and Magnenat-Thalmann, N., (1987), "Information Systems in Small Business: Are they used in Managerial Decisions?", American Journal of Small Business, Vol. 6, No 4, pp. 20-26

Robinson, R.B., Jr. (1982), "The Importance of 'Outsiders' in Small Firm Strategic Planning", Academy of Management Journal, Vol. 25, No 1, pp. 80-93

Salmela, H., and Ruohonen, M., (1992), "Aligning DSS development with organizational development", European Journal of Operational Research.

Sambamurthi, V., Venkataraman, S., DeSanctis, G., (1993), "The design of information technology planning systems for varying organizational contexts", European Journal of Information Systems, Vol. 2, No. 1, pp. 73-92.

Scholhammer, H., and Kuriloff, A.H., (1979), "Entrepreneurship and Small Business Management", New York: Wiley

Sengenberger, V., Loveman, G.W., and Piore, M.J., (1991), "The Remergence of Small Enterprises: Industrial Restructuring in Industrialized Countries", Geneva, Switzerland: International Institute of Labour Studies

Slevin, D.P., and Covin, J.G., (1990), "Juggling entrepreneurial style and organizational structure - How to get your act together", Sloan Management Review, pp. 43-53.

Strassmann, P.A., (1985), "Information payoff", Free Press, London.

Sutherland, A.R., and Galliers, R.D., (1989), "An evolutionary model to assist in the planning of strategic information systems and the management of the information systems function", Curtin University of Technology, Perth, Western Australia, February.

Triandis, H., and Vassiliou, V., (1972), "Interpersonal influence and employees selection in two cultures", Journal of Applied Psychology.

Veiga, J.F., and Yanouzas, J.N., (1991), "Differences between American and Greek managers in giving up control", Organization Studies, Vol. 12, No. 1, pp. 95-108.

Verber, B., and Zupancic, J., (1993), "Application of Information Technology in Small Business in Slovenia" In the Proceedings of the UK Systems Society Conference on Systems Science: Addressing Global Issues (Eds., F., Stowell, D., West, J., Howell), Plenum Press, pp. 493-498

West, G.M., (1975), "MIS in Small Companies, Journal of Systems Management", April, pp. 10-13

Wroe, B., (1987), "Successful Computing in Small Business", Manchester: NCC Publications

Yap, C.S., Soh, C.P.P., and Raman K.S., (1992), "Information Systems Success Factors in Small Business", OMEGA International Journal of Management Science, Vol. 20, No 5/6, pp. 597-609

DEVELOPMENT OF INFORMATION SYSTEMS TO SUPPORT ELECTRONIC COMMERCE

Jože Gričar

University of Maribor, School of Organizational Sciences
Prešernova 11, 64000 Kranj, Slovenia
internet: gricar@uni-lj.si

Abstract

Electronic Data Interchange - EDI is primary technology used to support electronic commerce. The real payoff from EDI may come from the business benefits resulting from trading partners working together to reduce costs and/or improve customer service. Main components of electronic commerce are internal organization applications of partners, EDI infrastructure, and common data base. Implementing electronic commerce will encourage organizations to examine their existing processes and to reengineer them to eliminate unnecessary steps and to automate those that remain. Electronic commerce will impact information systems development and information systems professionals should prepare for that.

EDI - ELECTRONIC DATA INTERCHANGE: THE FOUNDATION OF ELECTRONIC COMMERCE

Electronic Data Interchange - EDI is primary technology used to support electronic commerce (EC). A message to be exchanged is translated into standards-based transaction and sent to a partner via telecommunications, often via electronic mailbox.

EDI is not just any system which supports the exchange of business data or documents by electronic means rather than as paper documents or magnetic media. Some would use a term "Pre-EDI" for applications that exchange data by exchange of computer-readable media, such as magnetic tapes and disks. Even structured use of facsimile transmission may refer to this category.

Usually, EDI is defined as computer-to-computer exchange of routine business transactions (messages) between two (or more) organizations using standard formats. Message formats are referred to as transactions sets or electronic documents. Data are moved electronically, without any human intervention or rekeying. EDI may also be viewed as application-to-application exchange of messages. Data are sent from one computer application, such as a purchasing system of the buying company, to another computer application, such as an order processing system of the selling company. Translating software is built into either application systems or operating system. In near future EDI may be defined as person-to-person exchange of messages. Standards and telecommunications will be embedded into packaged systems and partners will just communicate one with another. In the multimedia environment also voice, charts and pictures will be transmitted in addition to data.

Basically, EDI is not new. Many of transactions already occur electronically but require prior arrangements and dedicated lines. In most cases, applications do not have the minimum capabilities, and gateways may have insufficient translation or communication facilities. The resulting costs and lead times create entry barriers to widespread participation by organizations, small business in particular. This hinders expansion of EDI beyond large companies and their major trading partners.

Standards are very important part of EDI. Widespread connectivity and interworking among diverse computing and communications systems is dependent upon the ability to plan and coordinate on a national and even global scale. This can be achieved only through the use of standards. Standards enable open systems. With respect to standards the term EDI is sometimes used in more restricted way, e.g. by limiting it to systems which use standards published by an industry, a national or international standards body, or which are provided by a recognized third-party EDI network services provider. Industry standards paved the way for EDI, for example TDCC (Transportation Data Coordinating Committee) in the US transportation industry, ODETTE (Organization for Data Interchange by Teletransmission in Europe) in the European automotive industry etc. EDIFACT (EDI For Administration, Commerce and Transport) is emerging international standard. As it holds for standards in general, it is true for the EDI standards as well that industry and national standards may be a barrier to global trade.

The use of EDI does not necessarily increase transactional risk beyond that experienced in a paper-based environment. Communications protocols, X.400 for example, contain data protection mechanisms. Viewed from the standpoint of potential threats, controls should make the cost of obtaining data greater than the potential value of obtaining or modifying data.

Implementation of EDI means a major change in an organization. One degree of change is, for example, replacement of a paper invoice by an electronic message. Quite another degree of change is complete avoidance of invoice, either paper or electronic one. In fact, the invoice is not needed since buyer knows payment terms agreed upon by order confirmation. If the buyer has receipt of accepted goods, why must the vendor request payment and why must the buyer handle invoice at all? Payment of goods on receipt without invoice is obviously an more efficient arrangement. Such an arrangement creates significant change in a business process, however.

EDI AND ELECTRONIC PARTNERSHIP

The real payoff from EDI may come from the business benefits resulting from trading partners working together to reduce costs and/or improve customer service. When two or more trading partners form an electronic partnership (Konsynski and McFarlan 1990) directed at improving their collective performance in the market place, and use EDI and other tools as facilitating mechanisms, wonderful things can happen. Perhaps most important of all is the attitude that trading partners adopt. If a supplier is at the end of the electronic leash of its most important customer because it has been told to use "EDI or DIE" that is one thing. If, on the other hand, customer and supplier have formed a partnership to make them collectively more successful by linking their operations electronically in an information partnership, that is another thing altogether (McCubbrey 1993). There is only one reason for commercial partnerships - profit (Jenkins 1994). Both parties must be better off than they would be without the partnership. Partnerships are neither inherently good or evil. They are simply artifacts of agreements between two or more organizations seeking to improve their profitability.

As always, economics is important also for EDI. EDI can make trade more efficient by reducing costs. However, cost savings are not the primary reason for information technology (IT) implementation. Improved customer service and improved information management are supposed to be the most important benefits of EDI. Costs are fixed and come early while savings are variable and come later. The most common reason for implementing EDI is simply a strong demand by customers. EDI is, however, only a tool, and there would be benefits in focusing on organizational processes rather than IT tools.

So far most of research in EDI has concentrated mainly on the examination of large and successful corporations that usually operate in well-established economies where formal structure and process are common. As a consequence all proposed models and systems in general are based on data collected from large organizations. On the other hand, in small and medium size enterprises (SMEs) information systems (IS) function is usually subordinated to the accounting function with limited expertise, experience and training in IS management. They are ineffective in deploying IS which can be attributed to the relative unawareness of the owner. Owners who may be quite effective in managing the business domain may lack the knowledge and miss the opportunity to exploit EDI equally well (Doukidis and Takis 1994). It should be stressed that several countries pay special attention to EDI implementation in SMEs. For example, Australia, Canada, Greece, European Union (TEDIS Programme). Since there are millions of SMEs on the globe here may be one more challenge for IS developers.

ELECTRONIC COMMERCE

Electronic commerce (EC) is paperless exchange of business messages using EDI, electronic mail, electronic bulletin boards, joint data bases and similar technologies. New aspects of electronic transactions are interactive EDI services in addition to traditional computer-to-computer exchanges and open market place in which transactions take place spontaneously on public networks in addition to proprietary ones. It is expected that EC will stimulate creation of massive, competitive vendor base.

As suggested by Clarke (1993) EC is an unifying or umbrella concept as a means of integrating the various electronic support technologies. The terms 'electronic trading' and 'electronic markets' have gained some degree of currency, but have the disadvantage of referring only to the agreement-making portion of the business process. 'Electronic business' used as an all-encompassing term covers many forms of service in which trading does not occur. The term 'electronic commerce' is a sufficiently broad notion, which can reasonably be used to encompass all of the trading, pre-trading and post-trading elements which need to be dealt with.

EC is a part of a broader information infrastructure in information society. Governments are more and more involved in creation and development of information infrastructure. For example, in October 1993 the President of US signed an Executive Memorandum (Memorandum 1993) requiring Federal agencies to implement the use of EC in Federal purchases as quickly as possible. Memorandum is a part of stately named US National Information Infrastructure.

Another governments initiative which is very relevant to EC is CALS (Continuous Acquisition and Life Cycle Support). CALS emerged initially in the world of US defense procurement. It is now expanding into the non-defense industry, not only in the US but also in Europe and Pacific Rim (Chalfont 1994). CALS is EDI and concurrent engineering. The vision is for all parts of a single enterprise, or, for example, an original equipment manufacturer and its suppliers, or a consortium of public and private groups to be able to work from a common data base, in real time, on the design, development, manufacturing, distribution and servicing of products.

EC may provide some answers to questions related to IT productivity paradox (Ives 1994). Productivity paradox proponents claim that investments in IT, though massive, have not produced significant improvements in industrial productivity. The issue is not in measuring IT productivity itself. Instead, it is understanding the impact of IT on the overall performance of activities. No longer is it enough to consider company-wide significance of IT and transcend divisional boundaries. Nowadays it is important to focus on long-term strategic plans and transcend organizational boundaries and even country borders. It is obvious that IS professionals should focus on the development of interorganizational information systems (IOS) as suggested by some authors several years ago (Kaufman 1966, Barret and Konsynski 1982, Johnston and Vitale 1988, McFarlan 1988).

ELECTRONIC COMMERCE ARCHITECTURE

Main components of EC are internal organization applications of partners, EDI infrastructure, and common data base. More specifically the following components are suggested by the US Federal Electronic Commerce Acquisition Team (Streamlining 1994):

◊ A single means of supplier registration to do business electronically including a standardized trading partner agreement. The agreement should be standardized at least at industry group level and embody the "rules of the road".

◊ Standardizing trade partner agreements can greatly reduce the time and expense to implement EC

◊ A standard method of implementing EDI using standard format

◊ Data in flat file format should be available for translation using commercial off-the-shelf software package to generate EDI messages. Modification of existing applications to generate standard EDI transactions will be required

◊ A "virtual network" connecting organization standardized transactions to facilities where value-added VANs or other entities can access them

◊ Standards-based system that gives staff access to organization data bases supporting their operations

◊ The use of electronic funds transfer (EFT) as the principal method of payment and the development of a supportive EFT architecture

◊ A standard agreement between the organizations and the VANs to support trading partners

◊ Security has to be provided for the system to process and pass information in a secure manner. Organizations must provide equivalent levels of security in EDI environment as they provide for paper

◊ Availability 24 hours a day, 7 days a week. Accessibility to the system may be required globally in international business.

◊ Costing and charge-back systems have to be defined in order to keep system transparent to the partners. Parties have to decide about payment for the movement and storage of data.

The above components of electronic commerce indicate directions that organizations doing business with the government will have to follow. These directions will have an impact on IS development in foreseeable future. IS professionals will have to consider how electronic commerce architecture may impact IS development.

We can see new promises of rapidly changing IT that require a change in our perceptions of IT. They require paradigm shift (Lalonde 1993; Tapscott and Caston 1993). IS professionals should be proactive in exploring new avenues of IT use and exploit its capabilities. The Society for Information Management - SIM, an international organization, is an example of a proactive approach. SIM recently published its vision and positions regarding the US National information infrastructure (Special Report 1994). The vision for exciting future is based upon an ever expanding ubiquitous network and associated applications. SIM defined eight positions towards The National Information Infrastructure. Three of them relate to EC issues: standards for connectivity and interoperability, access, and Internet.

◊ Standards for connectivity and interoperability. Lack of common standards today is the principal limiting factor for growth of the highway concept. Global standards should be incorporated where appropriate.

◊ Access. The government should be responsible to ensure universal accessibility to national information highway, however the accessibility to applications on the information infrastructure should be determined primarily by free market forces.

◊ Internet. SIM is grateful to Internet community for opening their network for commercial use, however concerned about Internet's ability to scale up both technologically and administratively. While Internet could serve as a very useful electronic post office, it should not, by default, become the only one.

Value Added Networks - VANs play an important role in development of EC. They reduce implementation activities, provide EDI capable partners, and educate trading partners. VANs also eliminate most of hardware/software compatibility problems and act as catalysts for promotion and maintenance of standards. An organization is usually connected to one VAN and VANs are responsible to provide connectivity among themselves. The development of X.400 (international standard for electronic mail based on OSI - Open Systems Interconnection), and X.500 (OSI international standard for directory services) will increase VANs interoperability and support user base growth in competitive environment.

SOME FUNCTIONAL REQUIREMENTS OF ELECTRONIC COMMERCE

For EC to be efficient and effective some functional requirements are suggested (Streamlining 1994):

- Demonstrate potential benefits of doing business electronically. Develop and present technological and organizational prototypes.

- Establish a data base of trading partner information. Use agreed upon unique suppliers numbering scheme. Organization should provide one face to its business partners.

- Minimize the need of trading partners to reprogram their systems or purchase different software to handle each partner's data structure. Implement standards. Develop standards-based open system.

- Develop and implement industry-wide (country-, world-wide) standard trading partner agreement. The standardized terms and agreements should be broad enough to accommodate the majority of the operating and procedural requirements.

- Streamline and standardize contract content to the maximum practicable extent. This includes standardization of clauses and provisions. Consider best commercial practices.

- Provide for organizations to have access to prescriptions and regulations associated with legal an business provisions and clauses.

- Provide a standard set of technical requirements and communication protocols and have a way of validating the technical competency of VANs.

- Provide for full electronic commerce capability that accommodates a range of business activities, including the ability for oral and written clarifications. In addition to EDI capabilities, EDI to fax, electronic mail, and voice mail should be available to accommodate all EDI-capable and non-EDI capable trading partners to allow full electronic capability.

Application systems should maximize the use of electronic commerce architecture. They must provide the end user with a standardized, transparent capability to exchange messages. Applications have to be structured to support both EDI and non-EDI data exchange.

EC architecture should provide the capability to exchange data within one-to-one relationship between two specific trading partners, within one-to-all relationship involving the distribution of messages to all interested parties, and within one-to-specific relationship involving the distribution of messages to selected parties.

Technical architecture must include a telecommunications capability that is efficient, reliable and capable to accommodate the anticipated increasing volume of EC traffic. Interfaces to EDI gateways and VANs, trading partners, and their applications are required. Systems should be able to process complex documents containing drawings, charts, technical specifications, and textual data.

SOME OBSTACLES TO EDI AND ELECTRONIC COMMERCE

In general, potential benefits of EDI and EC seem to be quite promising. However, there are also several obstacles to implementation of EDI and EC development:

- Business executives and IS professionals do not realize the potential power of EDI as an enabler of efficient and effective trading

- Established ways of thinking and viewing paper based systems as the only possible way of doing business prevent visualizing electronic trading

- Current policies and internal controls do not support implementation of new business procedures and techniques. For example, use of checks instead of credit or debit cards

- Technological and organizational barriers do not allow sharing common data and provide data base access

- Current application systems of an organization are usually not integrated. This is even more the case with application systems that support business between partners

- Existing legal or legislated arrangements do not promote EC

REENGINEERING OF BUSINESS PROCESSES - INTEGRAL PART OF ELECTRONIC COMMERCE DEVELOPMENT

Business process reengineering (redesign, renewal, reorganization, rationalization) is an important topic supported no longer just by several papers, but also by some much referred books (Davenport 1993, Hammer and Champy 1993). It is known as a hot management tool (Stewart 1993). Although there are some disagreements about how (non)radical reengineering can or should be (Davenport and Stoddard 1994), there is a general consensus that the use of IT enables and requires process reengineering. In this paper we refer to an impact of EDI on process reengineering.

EDI can be applied to, or used as the catalyst for reengineering existing processes (Bons et al. 1994, Bytheway 1994, Hofman 1993, Knoppers 1992, Roberts and Flight 1994, Swatman and Swatman 1993, Wagenaar 1992, Wrigley 1992, Vogel 1993). From a strategic standpoint, one should explore the potential of EDI as an enabling technology that supports streamlined business processing and facilitates electronic communications between organizations. The process of moving to a long-term strategy for EC implementation should include a more rigorous look at reengineering needs and building a system that takes advantages of the greatest number of improvements.

Basic business processes must change if we are to realize all the benefits of paperless operations. Focus will have to be placed on process simplification through the reworking of the paper oriented processes. Implementing EC will encourage organizations to examine their existing processes and to reengineer them to eliminate unnecessary steps and to automate those that remain. To integrate processes and data in the organizations with those of their customers, suppliers, other business partners, and governmental agencies we have to look beyond traditional organizational boundaries.

Processes may be reengineered at different levels. First level is an individual organization linking various processes of acquisition, buying, transporting, receiving, and paying for materials and services. Second level are several organizations involved in a certain business. We have to adopt a process view (value-chain approach) and a new business model of an extended enterprise. The broader the "system" that we imagine, the more benefits may be gained, however more effort and pain may be required.

As suggested by Vogel (1993), reengineering efforts make use of models which allow study teams to simulate or estimate the effectiveness of new arrangements as well as model existing organizational process. The model of an entire organization is a complex artifact that must combine the knowledge from people through the organization in a consistent representation. Multiple views have to be provided. Furthermore, it may not take too long to build a model to plan changes. Group process modeling become a key element in moving organizations forward in EDI. Group support systems were developed to support modeling.

GLOBAL IMPLICATIONS OF ELECTRONIC COMMERCE

Business process reengineering and IT implementation may be positioned into a broader, global trade perspective. Information infrastructure can increase trade efficiency, which allows faster, simpler, broader and less expensive trade. According to conservative estimates, the current cost of trade procedures (i.e. paper-work, licenses, inspections, etc.) roughly equals 10% of the value of international trade, i.e. over 300 billion US $ per year (Lanvin 1993). By its initiative of "trade points", UNCTAD - United Nations Conference on Trade and Development expects to cut the costs by 25% by the year 2000. This is expected to be achieved by simplification of business processes and procedures, and information infrastructure implemented in trade points.

Trade points are IT supported (EDI, telecommunications, and networks) entities dispersed world-wide geared especially to meet the needs of small and medium sized companies, within their geographic area in native languages. They will provide information on foreign trade procedures with practical information on all aspects required to carry out any foreign trade transaction. Provided information relate to customs regulations, duties and taxes, documentation, export/import restrictions, transport, forwarding, preshipment inspection, terms of payment, terms of delivery etc. Trade points should maintain data bases with information on available goods and services as well as potential buyers. The world markets are expected to come to everyone around the world, around the clock.

UNCTAD initiative is on the line of broader movements for IT implementation to support international trade. Just to mention European Union, NAFTA, and emerging new International Trade Organization, which will replace General Agreement on Tariffs and Trade - GATT by January 1995. The environment of IS development is becoming global indeed.

SUGGESTIONS

The following considerations can be proposed to IS developers:

⇒ Enlarge the scope of a "system". Include processes of as many partners as possible

⇒ Provide for executive support

⇒ Provide for sufficient understanding of business processes within IS professionals community

⇒ Provide for sufficient understanding of IT within user community

⇒ Include all stakeholders involved

⇒ Integrate EDI into reengineered work flow

⇒ Involve other professionals in systems development, for example lawyers, auditors etc.

⇒ Cooperate with VAN services providers

⇒ Estimate the tradeoffs between costs and efforts required against benefits expected

⇒ Consider international implications of doing business electronically

⇒ Prepare for adequate training.

REFERENCES:

Barret, S. and Konsynski B. R., Interorganizational Information Sharing Systems. MIS Quarterly, December 1982; 93-105.

Bons, R. W. H., Lee, R. M., Wagenaar, R. W. and Wrigley, C. D., Computer Added Design of Inter-organizational Trade Scenarios: A Case for Open EDI. Proceedings. The Seventh International EDI-IOS Conference: Electronic Commerce - Electronic Partnership. Bled, June 6-8, 1994; 180-193.

Bytheway, A., The Impact of EDI on Business Process Redesign: The Cranfield Studies. Proceedings. 5th World Congress of EDI Users. Brighton, UK, 14th-17th June 1994; 142-163.

Chalfont, Lord, Looking to the future. Proceedings. 5th World Congress of EDI Users. Brighton, UK, 14th-17th June 1994; 344-348.

Clarke, R. A., EDI is but one Element of Electronic Commerce. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 234-243.

Davenport, T. H., Process innovation: Reengineering work through information technology. Boston, Mass.: Harvard Business School Press, 1993.

Davenport, T. H. and Stoddard, D. B., Reengineering: Business Change of Mythic Proportions. MIS Quarterly 18(1994)2, June 1994; 121-127.

Doukidis, G. and Lyberas, T., The business and social applications of EDI on SMEs. Proceedings. 5th World Congress of EDI Users. Brighton, UK, 14th-17th June 1994; 308-319.

Hammer, M. and Champy, J., Re-engineering the corporation: A manifesto for business revolution. New York, N.Y.: Harper Business, 1993.

Hofman, W. J., Business Re-engineering: The Specification of IOS. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 171-185.

Ives, B., Probing the Productivity Paradox. Editor's Comments. MIS Quarterly 18(1994)2, June 1994; xxi-xxiv.

Jenkins, A. M., Electronic Commerce Means Partnership for Profit. Proceedings. The Seventh International EDI-IOS Conference: Electronic Commerce - Electronic Partnership. Bled, June 6-8 1994; 30-31.

Johnston, H. R. and Vitale, M. R., Creating Competitive Advantage with Interorganizational Information Systems. MIS Quarterly 12(1988)2; 153-165.

Kaufman, F., Data Systems that Cross Company Boundaries. Harvard Business Review, January-February 1966; 141-155.

Knoppers, J. V. Th., Importance of the "Open EDI" Reference Model from a User and Business Perspective. Proceedings. The Fifth EDI Conference: EDI Interorganizational Systems in the Global Environment. Bled, September 3-5 1992; 50-78.

Konsynski, B. R. and McFarlan, W. F., Information Partnership - Shared Data, Shared Scale. Harvard Business Review, September-October 1990; 114-120.

Lalonde, R., EDI, Part of a Larger Paradigm Shift. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 368-376.

Lanvin, B., Telecom essential for international trade. Transnational Data and Communications Report, January-February 1993; 12-14.

Memorandum for the Heads of Executive Departments and Agencies (and) the President's Management Council. Streamlining Procurement Through Electronic Commerce. Federal Register, Vol. 58, No. 207, October 28, 1993.

McCubbrey, D. J., The Benefits of Partnership: The Real Payoff from EDI. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 76-81.

McFarlan, W. F., Editor's Comments. MIS Quarterly 12(1988)2; ix-x.

Roberts, B. and Flight, G., The Enabling Role of EDI in Business Process Re-engineering. Proceedings. The Seventh International EDI-IOS Conference: Electronic Commerce - Electronic Partnership. Bled, June 6-8 1994; 232-239.

Special Report: Vision and Positions Regarding the US National Information Infrastructure. Society for Information Management - SIM, 1994.

Stewart, T. A., Reengineering - the hot new managing tool. Fortune, August 23, 1993; 41-48.

Streamlining Procurement through Electronic Commerce. Federal Electronic Commerce Acquisition Team. Review Draft. Fall Church, Virginia, April 1994.

Swatman, P. M. C. and Swatman, P. A., Business Process Redesign Using EDI - An Australian Success Story. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 116-137.

Tapscott, D. and Caston, A., Paradigm Shift: The New Promise of Information Technology. New York: McGraw-Hill, 1993.

Wagenaar, R. W., Business Network Redesign - Lessons from the Port of Rotterdam Simulation Game. Proceedings. The Fifth EDI Conference: EDI: Interorganizational Systems in the Global Environment. Bled, September 3-5 1992; 390-404.

Wrigley, C. D., EDI Transaction Protocols in International Trade. Proceedings. The Fifth EDI Conference: EDI: Interorganizational Systems in the Global Environment. Bled, September 3-5 1992; 405-418.

Vogel, D., EDI Group Process Modeling. Proceedings. The Sixth International EDI-IOS Conference: Strategic Systems in the Global Economy of the 90s. Bled, June 7-9 1993; 234-243.

Vogel, D., Business Process Improvement Through Electronic Commerce. Proceedings. The Seventh International EDI-IOS Conference: Electronic Commerce - Electronic Partnership. Bled, June 6-8 1994; 221-231.

Modelling

ANALYZING THE NOTIONS OF ATTRIBUTE, AGGREGATE, PART, AND MEMBER IN DATA/KNOWLEDGE MODELLING

Renate Motschnig-Pitrik

Institute of Applied Computer Science and Information Systems
Department of Data Engineering
Rathausstrasse 19/4
1010 Vienna, Austria

Abstract

One objective of conceptual modelling is to capture as much of the semantics of the real world as possible, in order to construct models that closely reflect some aspect of reality. Different techniques stemming from database modelling, software development, and AI have been proposed to approach this goal. Almost all of them apply the notions of attribute, aggregate, part, and member. The exact meaning of these notions, however, often remains concealed and differs from approach to approach. This is the case not only across individual disciplines of computer science but, as will be shown, even within particular paradigms, such as object-oriented modelling. This paper argues that an inconsistent use of fundamental concepts prohibits their effective application and leads to a decrease of the quality of the resulting software. The goal of this paper, therefore, is to clarify the terminological ambiguities by tracing and defining the semantics of each of the concepts mentioned. Further, the importance of their distinction will be analyzed and the benefits of their proper support will be assessed, irrespective of the choice of a particular technique or paradigm of development.

1. INTRODUCTION

Information Systems (IS) design strives to model some aspect of reality and represent it in an appropriate model. A number of models, semantic as well as object-oriented (OO), have been developed that aim at capturing and representing the semantics of real-world applications. In general, these models provide more expressive concepts with which to capture meaning than classical data models. An analogous trend can be observed in software development, where traditional development techniques such as SADT (Ross 1977) or JSD (Jackson 1982) give way to novel, OO analysis and design techniques that provide richer modelling constructs. These novel techniques are also better tuned for integration of the design of application programs with that of a database (Martin and Odell 1992).

Drawing on the work on semantic networks in Artificial Intelligence (AI) (Brachman 1979), semantic data models succeeded in proliferating powerful abstraction concepts, in particular classification, generalization/specialization, aggregation and, to some degree, association (the member-of relationship) (Hull and King 1987). Further, the concepts of object or entity, attribute, and relationship, which were originally defined in the context of the Entity-Relationship (ER) Model (Chen 1976), are broadly used in IS development.

The convergence of techniques stemming from different fields such as databases, software engineering, and AI towards the use of a common set of powerful modelling concepts has the potential of resulting in systems that are easier to understand, develop, integrate, and maintain. These benefits, however, can be achieved only if the high-level modelling concepts are understood and applied in a consistent way. Currently, different techniques tend to apply modelling concepts in an ambiguous, if not inconsistent way. We conjecture that this is mainly because the semantics of many modelling concepts have not been sufficiently understood such that they are constantly being reinterpreted in several ways. Surprisingly, inconsistent uses of semantic concepts appear not only across fields like data modelling and software development and across phases like conceptual modelling versus programming, but can be traced within individual disciplines and and even paradigms. As an example (see also Table 1) consider Rumbaugh et al.'s (Rumbaugh et al. 1991) Object Modelling Technique (OMT) and Embley et al.'s OO Structured Analysis (OSA). What is referred to as association in OMT is called relationship in OSA. Yet, association in OSA refers to an abstraction, in which a set of objects is considered a higher level set object. This abstraction is not explicitly supported in OMT, but is subsumed by the abstraction concept of aggregation, which is applied to model the part-of relationship. Further, the concept of an attribute is explicitly supported in OMT but subsumed by the relationship concept in OSA.

Table 1: Example of inconsistent uses of terms in OO modelling techniques

concept	OMT (Rumbaughet al. 1991)	OSA (Embley et al. 1992)
attribute	attribute	relationship
relationship	association	relationship
part-of *relationship*	part-of relationship synonymous with aggregation; denotes real-world parts only	is-part-of relationship synonymous with aggregation; denotes parts of an object's representation
member-of relationship	subsumed by part-of relationship	distinguished from is-part-of relationship

A further source of confusion regards conceptual modelling versus implementation concepts. As should become clear shortly, it is perfectly justified that different concepts from the conceptual modelling level (such as attributes, relationships, parts) are implemented using one concept only, say slot, field, instance variable or the like. This does not mean, however, that providing a variety of concepts at the conceptual modelling level is superfluous. Yet one has to be aware of the "overloading" of terms such that their semantics differ with respect to the level of specification.

Previous research has addressed semantic concepts such as different kinds of abstractions (Smith and Smith 1977; Brodie et al. 1984; Hull and King 1987), the intuitiveness of the ER Model (Goldstein and Storey 1990) and the meaning of semantic relationships (Storey 1993; Wand et al. 1993). This paper investigates the semantics of the concept of attribute, aggregate, part, and member aiming to contribute to a more coherent understanding and use of these concepts. The discussion is informal at many places being based primarily on examples. The objectives of this paper are to

- analyze the concepts mentioned above and reveal similarities and differences;
- introduce an unambiguous terminology and propose a preliminary taxonomy of semantic relationships;
- define the semantics of the individual concepts as rigorously as possible when making only few assumptions of an underlying data/knowledge model;
- examine the applications of the concepts in conceptual modelling as well as in the implementation phase with specific emphasis on OO approaches;
- investigate the benefits of distinguishing the four concepts at the level of conceptual modelling.

The paper is organized as follows: the next section investigates previous work on the original and derived uses of semantic modelling concepts and introduces some aspects of a *taxonomy of semantic interconnections*. The taxonomy underlies the definitions of terms proposed to be used for attribute, aggregate, part, and member and is intended to expose similarities as well as differences of these concepts. The third section focuses on the benefits gained from a distinction of attributes from relationships, whereas section four deals with the semantics of the part-of and member-of relationships, both from a cognitive and a modelling perspective. Furthermore, the property of transitivity is investigated and the modelling benefits of distinguishing between parts, members, and aggregates are examined. Section five discusses the results and concludes the paper.

2. TERMINOLOGY AND RELATED WORK

In the following, we trace the original meaning of the concepts of attribute, aggregate, the part-of relationship, and the member-of relationship. We proceed by investigating some of the derived interpretations of these concepts. Favoring the original term-meaning associations, a preliminary taxonomy on semantic interconnections will be proposed.

2.1 Original and Derived Interpretations of Attribute, Aggregate, Part, and Member

Attribute. The attribute concept has traditionally been used in the ER data model (Chen 1976). The ER approach has pragmatically proved to be a highly successful conceptual modelling technique which represents information in terms of entities, attributes, and relationships among entities. While entities are the principal objects about which information is collected, attributes are used to provide identifying and descriptive properties, such as name, weight, or color. Relationships are used to represent interconnections among one or more entities, such as `works-for` in `employee works-for company`. Attributes are clearly distinguished from relationships. The main difference is that *attributes* represent *intrinsic properties*, the value of which does not depend on other entities in the model, while *relationships* describe *mutual properties* causing interconnections. In the following, the terms attribute and relationship will be used precisely in the sense of the ER model.

Other interpretations of the attribute concept generalize its original meaning. In the relational data model, for example, all fields of a relation are referred to as "attributes", even in the case that they represent a mutual property (e.g. by a foreign key). The same applies to OO notations where the state information of an object is often referred to as the object's structural "attributes", as opposed to behavioral "attributes" that denote the operations or methods of an object. Note, that in both cases the "attributes" can be interpreted as components of the underlying data structure, whereas only some of them describe attributes of the real-world concept being modelled.

Aggregation. Originally introduced by Smith and Smith (1977), *aggregation* denotes the record structure and means a conglomeration of attributes as well as relationships that describe a 'whole' object or entity. As such, an aggregate precisely matches the notion of a relation in a relational data model or the description of an object via its structural "attributes". As an example, consider the data type specified in figure 1a to capture information about papers.

In the sequel, another variant of the aggregate concept has been distinguished. In semantic data modelling and some OO techniques, aggregation tends to be used in the sense of an abstraction in which related objects are combined to form a higher-level aggregate object. Note, that these two uses are intimately related and can be seen as two levels of aggregation (Codd 1979). On the first level, an object is 'assembled' from attributes and relationships denoting, respectively, intrinsic and mutual properties of some real-world object. On the second level, related objects are 'assembled' to form a higher-level object. Since the higher-level object, referred to as aggregate, may have new, *emergent* (Wand et al. 1993) properties, it again can be seen as an aggregate of the first kind, namely one consisting of attributes and relationships (with its constituent objects). Aggregation is frequently used to denote the part-of relationship. The synonymous use of these two terms constitutes another source of confusion, as is argued below.

```
TYPE paper
  WITH
        title: String
        length: 1..100
        author: Person
        abstract: Text
        figures: Image
end
```

Figure 1a: Example of an aggregation

```
TYPE paper
  WITH
  attribute
        title: String
        length: 1..100
  relationship
        author: Person
  part
        abstract: Text
        figures: Image
end.
```

1b: Example of distinguishing different categories of components in an aggregation

The part-of relationship. By definition, *aggregation* serves to assemble related objects and/or attributes to describe a 'whole' or a higher-level object. Undoubtedly, the individual constituents of an aggregate, be it related objects or attributes, constitute components of the representation (for example a record type as in figure 1a) of an object (type). In this role, they are parts of an object's representation (specified, for example, in a type declaration). This does not mean, however, that the components of the representation also denote parts of the real-world object to be modelled. In general, only some of the representational components correspond to parts of the underlying real-world object. For example, `abstract` and `figures` are parts of a `paper`, whereas `length` is a property but not a part of a `paper` (although it is a component of the representation of the type `Paper`). This situation is illustrated in figure 1b that shows a distinction of the representational components of `Paper` into attributes, referential relationships, and parts. Note, that the ER model distinguishes between attributes and relationships but does not explicitly distinguish relationships denoting the part-of interconnection. This ambiguity regarding the part-of relationship leads us to differentiate between a representational and a real-world interpretation of the *part-of* relationship. Since the real-world part-of relationship carries important additional semantics (compare section 4), we follow (Kim et al. 1989) in proposing to use the term part-of *only* to refer to the real-world interpretation, while the representational interpretation more appropriately will be called (representational) component-of (compare figure 3) having as synonyms terms like field, slot, state function, or instance variable. Thus, for example, instead of saying `author part-of paper`, we will say `author component-of paper`, meaning that the representation of the `paper` concept has `author` as one of its components, irrespective of the real world semantics underlying the interconnection between `author` and `paper`. By the same token, `length` is a component-of the representation of the paper concept.

The member-of relationship. Traditionally, the *member-of* relationship is used in set theory to denote the relationship between a set and its members. Brodie (Brodie 1981) introduced many aspects of the modelling of sets into semantic data models by extending the latter by a set modelling construct called association or grouping. Brodie's association is an abstraction in which a relationship between member objects is considered as a higher-level set object. This relationship is called member-of and is distinctly distinguished from the part of relationship. Thus, for example, a person is member-of a club, while an engine is part-of a car.

In the context of the member-of relationship there are two sources of confusion. Firstly, many development techniques do not distinguish between the part-of and the member-of relationship but take part-of to subsume member-of. Second, the term association is frequently applied to denote any arbitrary relationship instead of denoting a set-based abstraction. To avoid ambiguity, we suggest to prefer the term grouping or collection to designate a set-based abstraction. A discussion on the benefits of distinguishing between the part-of and the member-of relationship is postponed until section four.

2.2 Towards a Taxonomy of Semantic Relationships

So far we have recalled the traditional and some of the derived uses of the four semantic concepts under discussion. In order to be able to rigorously argue on their meaning, their similarities, differences, and the benefits of their distinction, we need to introduce an unambiguous terminology. This will be done first by classifying properties of conceptual entities or objects[1] as conceived in the real-world and second by defining how the resulting property categories are mapped into modelling concepts. Figure 2 shows one level of a taxonomy of properties of conceptual entities. The taxonomy combines ontological findings with those of cognitive psychology. The distinction between intrinsic and mutual properties has been propagated, for example, by (Bunge 1977) and (Wand et al. 1993), whereas the importance of organization and structure to aid understandability has been researched, for example, by (Anderson 1990) and (Motschnig-Pitrik 1990).

In figure 2, properties are classified to belong to one of three categories[2]:

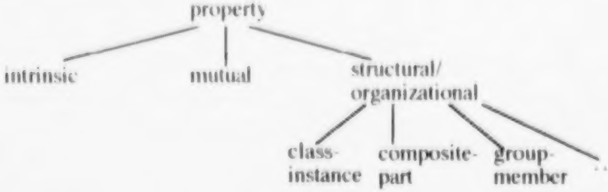


Figure 2: Classification of properties

Intrinsic properties are characterized by depending on one entity only such that they describe that one entity. Typical examples are the height of a person or the color of a car. Formally, intrinsic properties describe an entity by associating it with a value or an atomic object in the case of OO languages that do not distinguish objects and values, such as Smalltalk (Goldberg and Robson 1983).

Mutual properties depend on more than one entity and describe an application-specific interconnection between entities. As an example consider an employee being assigned to a project. in this case, the employee entity depends on the project entity and vice versa. Formally, mutual properties describe an entity by associating it with other (non-atomic) entities (not necessarily having different entity types).

Structural or *organizational properties* describe general-purpose interconnections among lower-level and higher-level entities supporting the organization of complexity via abstractions. As an example consider the entity type `Person` as an abstraction of individual people.such as you and your best friend or a club as an abstraction of club members. Formally, an organizational property describes an entity by associating it with a higher-level, more abstract entity, or vice versa. The higher-level entity may be concrete or abstract, having existence in the real-world or the human conceptualization thereof.

After having identified categories of properties as they are conceived in the real world, we have to specify how these categories correspond to modelling concepts. This is demonstrated in figure 3, which shows some aspects of a taxonomy of semantic interconnections and their mapping into modelling concepts. In particular, figure 3 shows the notion of an attribute as the denotation of an intrinsic property and part-of and member-of as special cases of organizational relationships. In contrast, the notion of a (representational) component is introduced to denote any kind of property and thus any kind of interconnection subsuming attributes and relationships. In this context, the notion of an *aggregate* is defined as an entity associated with *components*. Note that this definition subsumes the two levels of aggregates discussed in section 2. Note further, that this definition of an aggregate built from components is more general than the notion of a composite object built from objects interconnected by the part-of relationship.

---

[1] The term object is used informally and interchangeably with (conceptual) entity in the context of this paper.

[2] Note that properties of relationships may be categorized in a similar way except that mutual properties do not have significance.

The taxonomy shown in figure 3 depicts semantic interconnections between objects. An analogous taxonomy could be constructed to apply to the *type level* by substituting 'object' by 'object type' and 'value' by 'value type'. The only difference between the object and type level regards the organizational relationships: the type level can be extended by the *is-a* relationship specifying the generalization of types. The interconnection between the object and the type level is established by the *instance-of* relationship that equally falls into the category of organizational relationships.

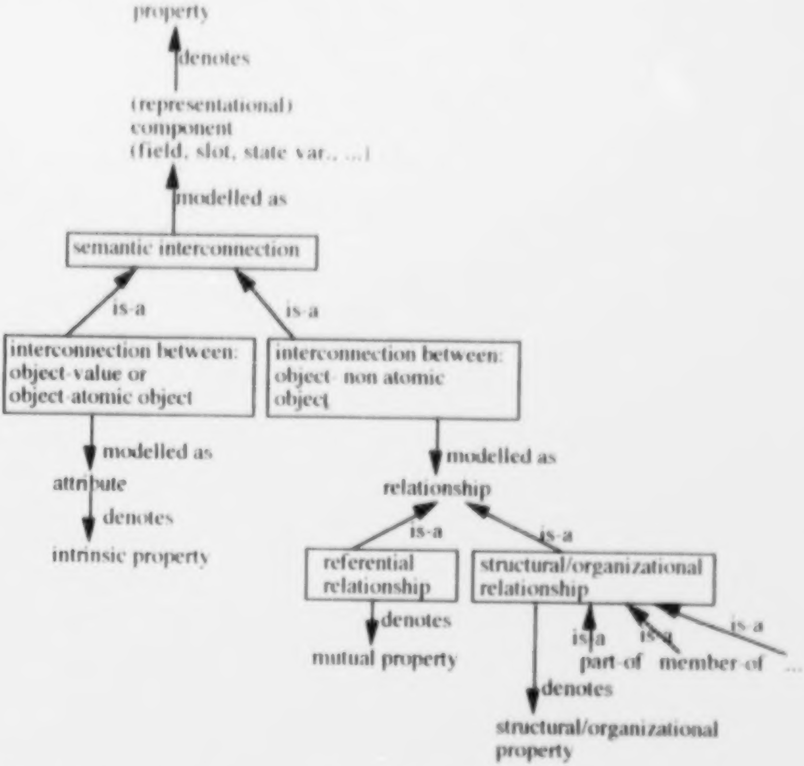

property
denotes
(representational)
component
(field, slot, state var., ...)
modelled as
semantic interconnection
is-a
is-a
interconnection between:
object-value or
object-atomic object
interconnection between:
object- non atomic
object
modelled as
attribute
denotes
intrinsic property
modelled as
relationship
is-a
is-a
referential
relationship
structural/organizational
relationship
denotes
mutual property
is-a
is-a
is-a
part-of
member-of
...
denotes
structural/organizational
property

Figure 3: Preliminary taxonomy of semantic relationships

BEST COPY AVAILABLE

3. DISTINGUISHING ATTRIBUTES FROM RELATIONSHIPS

Aside of the conceptual and ontological differences between attributes and relationships, there exist important pragmatic reasons why it is advantageous to distinguish these notions in conceptual modelling. Consider first the traditional way of database design that employs an ER-like model for conceptual design and transforms this model to a classical, say relational, model at the logical level of design. In ER-modelling it has been observed (see, for example (Teorey et al. 1986), (Goldstein and Storey 1990) p. 21,22) that if relationships are modelled as attributes, this often violates normalization and leads to inefficiency. Considering the representation of `Paper` shown in figure 1, a failure to model authorship as a relationship and using an attribute instead would violate normalization in the case of multiple authorship. Another aspect regarding the benefits of distinguishing attributes from relationships is illustrated by an example shown in figure 4 that models the assignment of employees to projects. Given that only some employees are assigned to projects, modelling `Project` as an attribute of `Employee` means that storage for it has to be allocated in each `Employee` record (although the value will be null in many cases) which may be quite inefficient. Also, a query to find all employees who work on a given project requires the traversal of the whole `Employee` relation. If, on the other hand, the interconnection between `Employee` and `Project` is modelled as a relationship (as shown in figure 4), the database designer can choose a representation which requires less storage and allows for more efficient queries and smooth normalization. Furthermore, this representation is easier to extend in the case that one wishes to record attributes of the relationship like, for example, from which date to which date an employee was assigned to a project.

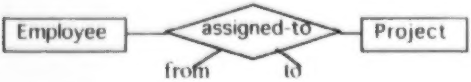


Figure 4: Example of a relationship denoting a mutual property

One might wonder whether there are benefits of distinguishing attributes from relationships when using an OO development method, since, in OO modelling, finally all information is encoded in an object that encapsulates information about state and behavior. In fact, the situation in OO modelling is almost symmetric to that in traditional database design, the key issue being what object types or classes (instead of relations in the classical approach) to introduce. If, for example, the relationship between `Employee` and `Project` were modelled as an attribute assigned-to with the range `Project`, this would be implemented as a pointer in `Employee`. It is easy to see that the arguments regarding inefficiency and bad extensibility apply exactly as it was the case above

In OO modelling, in particular, there exist further arguments on why attributes shall be distinguished from relationships. As noted in (Rumbaugh et al. 1991): "Implementing associations[3] as pointers is perfectly acceptable, but associations should not be modelled this way. ... Modelling a link as a pointer disguises the fact that the link is not part of either object but depends on both of them together. ... Furthermore, using a pair of matched pointers ... hides the fact that the forward and inverse pointers depend on each other." This implies the danger that one direction is forgotten in the case of an update.

In brief, distinguishing relationships from attributes at the conceptual level is important, since the former may be implemented in three ways. Which way to choose is a design issue. If, for example, a relationship is traversed in one direction only, it may be implemented as a pointer included in the source object. A one-to one relationship (link) that is traversed in both directions may be implemented as a pair of pointers, one in the source, one in the target object. Yet, a many-to-many relationship, such as the assignment of employees to projects, will tend to be implemented as a separate object type, say, `ProjectAssignment`. The latter case is particularly amenable to extensions, such as the addition of the date attributes `to` and `from`, used to record when an employee was assigned to a project.

---

[3] In [Rumbaugh91] the term association is used in the sense of a relationship.

Summarizing, the fact whether a property is intrinsic or mutual carries essential semantic information that deserves to be preserved in models of reality. Modelling this semantic aspect will be paid-off by an increased quality of the resulting software.

4. DISTINGUISHING PARTS AND MEMBERS FROM OTHER AGGREGATED COMPONENTS

In section two, it has been argued that the term *part-of* can be interpreted in two ways, one meaning part-of the representation of an object, the other meaning part-of some real-world object. It was suggested to apply the term part-of in the latter sense only and use the more general term component -of, or one of its synonyms such as field, slot, state-variable, in the context of an object's representation. This distinction can be justified from two complementary perspectives: cognitive science and semantic enrichment via constraints.

In a psychological experiment conducted by Tversky (Tversky and Hemenway 1984) subjects were asked to list "attributes" of object categories such as chair or bird. One result was that, on the average, more than 50% of the "attributes" listed were parts[4]. This result can be interpreted to prove that the relationship associating parts with composites is one of the most fundamental structuring concepts underlying the organization of human knowledge.

From a modelling perspective, parts carry specific semantics not shared by arbitrary relationships. As first observed by Kim et al. (1989), it is common for objects in OODB's to reference any number of other objects, but no specific semantics are captured by such reference links. The authors therefore suggest to superimpose the is-part-of relation on nested objects[5] such, that an object may be part of another object. A set of component objects which form a single conceptual entity (a whole) is then referred to as composite object and the links connecting the components with this object are called composite (or, in our terminology, part-of) links. Importantly, the model allows to specify for each part-of link whether the reference is *exclusive*, i.e. the component exclusively belongs to the composite, or *shared*, meaning that the component may possibly be part-of several composites. Further, a part-of link can be defined to be either *dependent*, which means that the existence of the component depends on the existence of the composite, or *independent*, i.e. having existence irrespectively of the composite. On the whole, four types of part-of relationships result from combining the two features.

Such additional semantics of the part-of relationship can be incorporated in data/knowledge models via formal constraints that must hold if objects are inserted, updated or deleted. As an example, consider the deletion of an object O':

> If there exists a dependent and exclusive part-of relationship from O' to another object O (i.e., O part-of O'), then it holds that the deletion of O' implies the deletion of O. In case that the reference is independent, however, the deletion of O' does not imply the deletion of O.

In (Motschnig-Pitrik 1993) it is shown how cardinality constraints can be inferred from the individual categories of part-of relationships. Figure 5 shows an example in which a paper is specified to be an exclusive and independent part of a journal. The fact that it is an exclusive part implies a maximum cardinality of one on the left side of the figure, whereas the fact that it is an independent part implies a minimum cardinality of zero. The cardinality constraints on the right side of the figure are not affected by the categories of the part-of relationship.

---

[4] The following definition was used to distinguish part terms: " A part is one of the segments or portions into which something is regarded as devided; a part is less than a whole; together, parts constitute a whole."

[5] The term nested object corresponds to an aggregate on the second level (see also section two).

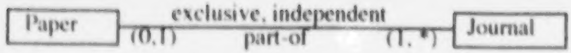


Figure 5: Example of part-of categories and their implication on cardinality constraints

Besides the part-of relationship, there exists another organizational relationship, namely member-of that appears generally applicable and carries specific semantics such that it is worthwhile to be distinguished from other relationships. The member-of relationship has been introduced by Brodie (Brodie 1981) to form an abstraction referred to as association. This abstraction views a set of homogeneous objects as a higher-level association object by abstracting from the properties of the individual member objects. As an example, figure 6 shows an association of students to form a club. This is accomplished by interconnecting the member type Student with the set (association) type Club by the member of relationship.

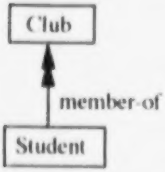


Figure 6: Example of a member -of relationship

Specifying `Student member-of Club` carries the semantics that each instance of the member type (Student) must be member-of at least one instance of the set type (Club). In other words, Brodie's version of the member-of relationship entails the constraint that set type instances must provide a cover of the member type.

A set modelling construct that forms a generalization of Brodie's association abstraction is proposed in (Motschnig-Pitrik and Storey 1994). The authors call it *grouping* in order to avoid confusion with recent uses of the term association as a synonym to relationship. The generalization affects mainly two dimensions. First, groupings may hold members of different types such that heterogeneous sets can be modelled directly. Second, it is not required that instances of the grouping type, i.e. groupings such as concrete clubs, provide a cover of the member type (Student). Instead, one of three general (predefined) constraints may be attached to the member-of relationship, in the case that such constraints reflect the real-world situation.

The ExclusiveMembers constraint demands that each instance of the member type(s) may be a member of at most one grouping. As an example consider a person being member of at most one political party. Next, the CoverMembers constraint reflects the fact that each instance of the member type(s) must be a member of at least one grouping. Consider, for example that a conference participant must be member of at least one working group. The third constraint capturing grouping semantics is called PartitionMembers and requires each instance of the member type(s) to be member of exactly one instance of the grouping type, such as an employee being member of exactly one department.

First note that in terms of the categories of parts discussed above, members of groupings can always be thought of as shared and independent parts. This is because the existence of member type instances does not depend on the existence of some grouping. Also, a member type can always be in member-of relationship with multiple grouping typees. For example, exclusive membership in one grouping, such as a political party, does not constrain member type instances to participate in other groupings, such as a club. This situation differs from that of exclusive parts that may be in part-of relationship with at most one composite. This is illustrated in figure 7 which shows that in case a wheel is specified to be an exclusive part of a car it may not be part of anything else.

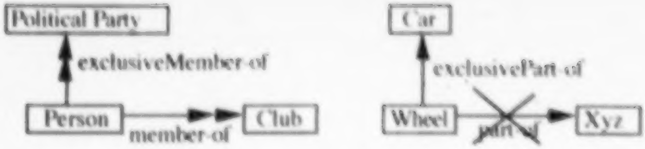


Figure 7: Example showing the difference between exclusive parts and exclusive members

So far we have argued that part-of and member-of are organizational, general purpose relationships such that each can be characterized by a set of constraints that are capable of capturing more of the real-world semantics than are inherent in other referential relationships. In many modelling approaches, however, the member-of relationship is not explicitly distinguished from the part-of relationship, this is despite the fact that cognitive and linguistic studies have proved that humans can readily tell the two relationships apart. In one representative experiment reported in (Winston et al. 1987), the authors had subjects sort word pairs such as forest-tree, cat-tail, etc.. This experiment showed that people not only distinguish part-like (meronymic) relationships from is-a-like ones. They are also capable of distinguishing member-of from other meronymic relationships such as those between objects and their components.

Transitivity. A further difference between part-of and member-of concerns the transitivity property. In general, a relationship R is transitive over the domain D of objects, if for any three objects A, B, C ∈ D it holds that

A R B and B R C --> A R C .

Transitivity is an important property in reasoning since it can be exploited for automatic inferences. For example, the transitivity of the is-a relationship is used by inheritance mechanisms. In the literature there exist dissenting opinions on whether the part-of relationship is transitive. Whereas, for example, Rumbaugh et al. (Rumbaugh et al. 1991) take the transitivity of the part-of relationship (which, in their case, subsumes the member-of relationship) for granted, Winston et al. (1987) argue that transitivity holds only within restricted categories of meronymic (i. e. part-whole) relationships, such as component - object, portion - mass, or group - member. In a semantic analysis of the part-of relationship (Motschnig.Pitrik 1993) it has been argued and illustrated by several examples that, given some precautions are taken, the part-of relationship is transitive whereas the member-of relationship is not. The latter can easily be proven by giving an example in which transitivity does not hold. For this reason consider the two premisses:

Book member-of Library;
Library member-of Social-Institutions.

These do not imply the conclusion:

Book member-of Social-Institutions.

Also, combinations of phrases containing part-of and member-of relationships tend to be non-transitive. As an example consider:

Warsaw part-of Poland;
Poland member-of United Nations.

These do not imply the conclusion:

Warsaw member-of United Nations.

As an example illustrating (but not proving) the transitivity of (pure) part-of, consider:
Abstract part-of Paper;
Paper part-of Journal
--> Abstract part-of Journal.

5. CONCLUSION

This paper was aimed at describing the widely but also quite ambiguously applied notions of attribute, aggregate, part and member. Rather than focussing on formal definitions, which are hard to accomplish without introducing a particular data model, our approach was based on examples from ER and OO modelling.

One major result was that distinguishing the notions of attributes and relationships at the conceptual level of modelling improves the quality of the resulting software. This can be attributed to the fact that attributes and relationships capture the semantics of different notions in the real world, namely those of intrinsic versus mutual properties. Almost as a side-effect it appeared essential to clearly distinguish conceptual modelling from implementation concepts. Lack of this distinction is apt to lead to misunderstandings, in particular, if a one-to-one correspondence of modelling and implementation concepts is assumed. A further important distinction concerns the real-world situation and its denotation in form of the data-structures provided by a model. In this respect it has been argued that distinguishing parts of a (real-world) composite object from the representational components (often referred to as parts) of the data-structure used to model that object allows additional semantics of the part-of relationship to be captured in the model. The resulting model then more accurately and faithfully reflects the real-world semantics. Not surprisingly, the advantages of distinguishing the individual semantic notions are obtained irrespective of whether traditional, ER-based, or more recent object-oriented development techniques are applied.

REFERENCES

Anderson J.: "Cognitive Psychology and its Implications"; Freeman, 1990.

Borgida A., Mylopoulos J., Wong H.K.T.: "Generalization/Specialization as a Basis for Software Specification"; in: "On Conceptual Modelling"; Brodie M.L., Mylopoulos J., Schmidt J.W., editors, Springer-Verlag 1984.

Brachman R.: "On the Epistemological Status of Semantic Networks", in *Associative Networks: Representation and Use of Knowledge by Computers*, Findler, N., V., (ed.), New York, Academic

Brodie M.L.: "Association: A Database Abstraction for Semantic Modelling"; Entity -Relationship Approach to Information Modelling and Analysis, P.P.Chen, editor, ER Institute, 1981.

Brodie M.L., Mylopoulos J., Schmidt J.W., editors: "On Conceptual Modelling"; Springer, 1984.

Bunge M.: "Treatise on Basic Philosophy: Vol.3: Ontology I: The Furniture of the World", Reidel, Boston 1977.

Chen P.P.S., "The Entity-Relationship Model - Towards a Unified View of Data"; ACM TODS, Vol.1, No.1, 9-36, 1976.

Codd E.,F.: "Extending the Database Relational Model to Capture More Meaning"; ACM TODS, Vol.4, No.4, 397-434, December 1979.

Embley D., W., Kurtz B., D., Woodfield S., N.: "Object-Oriented Systems Analysis - A Model-Driven Approach", Yourdon Press, Englewood Cliffs, NJ, 1992.

Goldstein R., Storey V., C.: "Some Findings on the Intuitiveness of Entity Relationship Constructs"; in: Entity-Relationship Approach to Database Design and Querying, F. H. Lochovsky, ed., Elsevier Science Publishers B.V. (North Holland), -23, 1990.

Goldberg, A. and Robson, D., Smalltalk-80: *The Language and its Implementation*, Addison-Wesley, 1983.

Hull, R. and King, R., "Semantic Database Modelling: Survey, Applications and Research Issues", *ACM Computing Surveys 19(3)*, September 1987.

Jackson, M., "System Develpoment"; Prentice Hall, 1982.

Kim, W., Bertino, E., Garza, J., F., "Composite Objects Revisited"; OOPSLA 89', 337-347, 1989.

Martin J., Odell J.: "Object-Oriented Analysis and Design", Prentice Hall, Englewood Cliffs, NJ, 1992.

Motschnig-Pitrik R., "A Framework for the Support of a Common Structural Level for Software, Database-, and Knowledge Based Systems"; The Journal of Systems and Software, North Holland, Vol.12, No.12, 157-165, 1990.

Motschnig-Pitrik R., "Toward a Common Structural Level for Software, Database-, and Knowledge Based Systems", *Applied Artificial Intelligence, An Int. Journal*, Hemisphere Publ., Vol.3, No.4, 405-426, 1991.

Motschnig-Pitrik R., Mylopoulos, J., "Classes and Instances"; Int. Journal on Intelligent and Cooperative Information Systems, Vol 1, No 1, World Scientific, 61-92, 1992.

Motschnig-Pitrik R.: "The Semantics of Parts Versus Aggregates in Data/Knowledge Modelling", Proceedings of the CAiSE'93, Lecture Notes in Computer Science, Springer, 1993.

Motschnig-Pitrik R., Storey V., C.: "The Grouping: an Effective Construct for Modelling Sets", submitted for publication, University of Rochester, N.Y., 1994.

Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas"; IEEE TSE Vol. SE-3, No.1, 1977.

Rumbaugh J., Blaha M., Premerlani W., Eddy F., Lorensen W., "Object-Oriented Modeling and Design", Prentice Hall, Englewood Cliffs NJ, 1991.

Storey, V., "Understanding Semantic Relationships", Very Large Database Journal, Vol 2, No.4, 455-488, October 1993.

Smith J., Smith D.: "Data Abstractions: Aggregation and Generalization", TODS, Vol.2, No.2, p. 105-133, June 1977.

Teorey, T., J., Yang, D., Fry, J., P., "A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model", ACM Computing Surveys, Vol.18, No.2, 197-222, June 1986.

Tversky B., Hemenway K.,: "Objects, Parts, and Categories", *Journal of Experimental Psychology: General*, Vol.113, No.2, 169-191, June 1984.

Wand Y., Storey V.,C., Weber R.: "Analyzing the Meaning of a Relationship"; Working Paper 92-MIS-001, Fac. of Commerce and Business Administration, The University of British Columbia, Vancouver, 1993.

Winston M. E., Chaffin R., Herrmann D.: "A Taxonomy of Part-Whole Relations"; *Cognitive Science*, Vol.11, 417-444, 1987.

ENTITY-RELATIONSHIP MODELLING OF INFORMATION SYSTEMS WITH DEDUCTIVE CAPABILITIES

Otto Rauh[1] and Eberhard Stickel[2]

[1] Fachhochschule Heilbronn, Daimlerstr. 35, D-74653 Künzelsau, Germany
[2] Europa-Universität Viadrina, Große Scharrnstr. 59, D-15230 Frankfurt (Oder)

Abstract

Today traditional information systems and deductive systems are modelled and implemented separately. Most users, however, demand integrated systems which meet their everyday information needs and have deductive capabilities as well. The well-known Entity-Relationship approach to conceptual data modelling is extended to cover modelling of such systems. First, a descriptive query language is introduced as a basis. Afterwards it is shown how this language may be used to define derivable schema components. In addition, a classification of derivable data is given which allows the analyst to decide which derivable data to include in the conceptual schema. Finally, the advantages and limits of the approach are discussed.

1 Introduction

Within only 15 years, the Entity-Relationship approach to data modelling, which is based on the Entity-Relationship Model, or ERM (Chen 1976), has become one of the most important methods for analyzing and modelling information systems, perhaps the most important at all in business data processing. Most of the systems modelled with ERM are traditional in the sense that they do not allow to store knowledge or make deductions from the data in the database. This does not mean that there is no need for systems with such capabilities. But as ERM does not offer any possibilities to describe knowledge up to now, deductive information systems are usually modelled with specialized languages such as predicate logic, frames or semantic nets (cf. Rich et al. 1991). This specialization is justifiable as long as distinctive deductive systems are concerned, consisting of many rules but only a few data, of rules that afford special evaluation techniques which are normally not available in database systems, or rules restricted to a small subject. The 'normal' user of business information systems, however, needs deductive capabilities which are comparably moderate, but he needs them in the context of his everyday information system.

The obvious thing to do is to enrich the ERM by adding means to define *derivable data*, at least to the extent they are needed by the users of business information systems and, at the same time, as far as they are manageable by the common relational database systems. Derivable data are data which are inferred (or deduced) from other data by applying a *derivation rule*. Data which are not deducible from other data are called *original*. In an ER schema, derivable data are represented by derivable schema components, i e derivable attributes, entity sets or relationship sets

The aim of this paper is to propose an extension to ERM which allows modelling integrated information systems with deductive capabilities, like those that are needed in business data processing We shall concentrate on the modelling aspects and not go into the details of implementation We ought to mention, however, that the information systems modelled with our approach, can be implemented using a relational database system if only non-recursive derivation rules are used

We will proceed as follows First, a simple language for the description of the structure of ER schemata is introduced Second, a query language is proposed which forms the basis for expressing the meaning of derivable data Then we show how a slightly enriched version of this language may be used to assign a meaning to derivable data We then introduce a classification of derivable data that may be used to decide whether certain data should be included in the conceptual schema or not Finally, we give an overview of the advantages and the limits of the approach

2 Description of Schema Structure

Figure 1 shows the ER diagram of a small personnel information system which will serve as the basis for our examples It can be easily understood without any explanations, apart from the constructs in dotted lines which are derived components and will be explained later We do not claim that such a scanty system is a realistic model of any enterprise, but the problems raised in the examples occur in practice, although they are likely to be formulated a little bit different in a real firm

As an ER diagram cannot show all the information about a schema, we give an additional verbal description in a schema declaration language (Figure 2) In the declaration of an entity set, we can see its attributes, which have been omitted from the diagram for the sake of clarity, and its identifier Declarations of relationship sets include the names of the participating entity sets, their roles within the relationship set, and the cardinalities Furthermore, there are the attributes, if any

The schema ist not quite complete, however First, the domains of the attributes are missing as they are not relevant for what follows Another missing part concerns derivable components Although these components are declared in the schema, they cannot be distinguished from original components We shall add some extensions to the schema description later which will provide this distinction

3 Description of Derivable Data

3.1 ERC: A Query Language for the ERM

We have already mentioned that we need an ER query language to define derivable data There have been several proposals for such a language, the first one being the fragmentary examples in Chen's original paper (Chen 1976) Other approaches are due to Zaniolo (1982), Markowitz et al (1982), Elmasri and Wiederhold (1982), and later Parent et al (1990), Lagrange (1990) and Hohenstein (1990) All these languages have in common that the result of a query is not an arbitrary entity or a relationship set Most of them produce output in form of tables like a relational query language Thus they are not strictly closed in the sense that the result of a query must have the same structures as the input data. The motivation behind the claim for closure is that in a closed language the result of operations may be used as input for other operations. As far as queries alone are taken into consideration, the missing closure property does not hurt too much, because often the connection between the result table and the components of the schema can be achieved by comparing attribute values. But if we want to use the query language as a basis for defining derivable components this approach is not sufficient. We want to embed the derivable components in the schema and show their connections to the other schema components. We do *not* want to show them as an additional relational database.

The query language we shall introduce now has its origin in the well-known relational tuple calculus, or TRC (cf. Ullman 1988 for a comprehensive discussion). We call it *Entity-Relationship Calculus* or *ERC*, for short. ERC may be used in the tradition of the ER languages mentioned above, that means with a table as the (always printable) result of a query, but there are also extensions which allow a query to have an entity set or a relationship set as the result. Now we will show the basic features of ERC, the extensions are discussed in the next section.

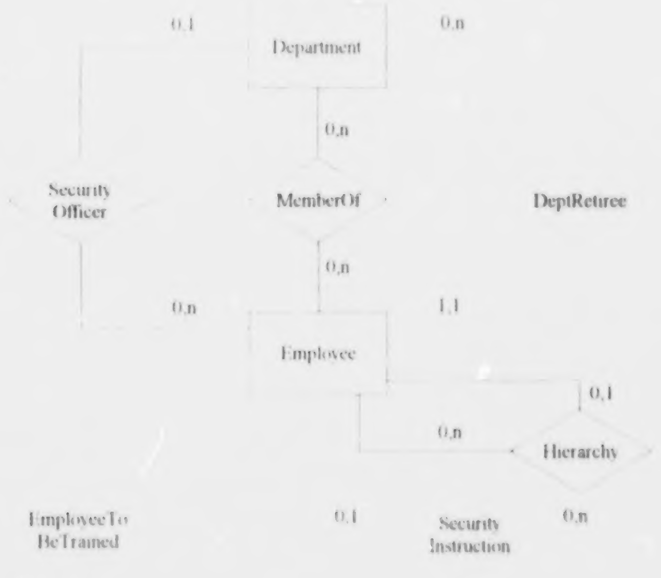


Figure 1. ER diagram

SCHEMA Personnel;

ENTITYSET Employee (ATTRIBUTES: EmpId, Lastname, Forename, Status, DateOfLastFirstAidCourse, IDENTIFIER: EmpId);

ENTITYSET Department (ATTRIBUTES: DeptId, DeptName, AvgTimeSpent, IDENTIFIER DeptId);

ENTITYSET EmployeeToBeTrained (SUBSETOF Employee, ATTRIBUTES: Delay);

RELATIONSHIPSET MemberOf (PARTICIPANTS: (Employee, Employee, (0,n)), (Department, Department, (0,n)), ATTRIBUTES: From, To);

RELATIONSHIPSET SecurityOfficer (PARTICIPANTS: (Employee, , Employee, (0,n)), (Department, Department, (0,1)));

RELATIONSHIPSET Hierarchy (PARTICIPANTS: (Employee, Superior, (0,n)), (Employee, Subalternate, (0,1)));

RELATIONSHIPSET DeptRetiree (PARTICIPANTS: (Employee, , Employee, (1,1)), (Department, Department, (0,n)));

RELATIONSHIPSET SecurityInstruction (PARTICIPANTS: (Employee, Trainer, (0,n)), (Employee, Trainee, (0,1)))

Figure 2 ER schema, verbal description

Let us describe ERC by means of some examples. They will be sufficient to give the general idea.

1) Lastnames and ID's of all security officers with the department ID's of the departments they have to care for

$$\{e.Lastname, e.EmpId, d.DeptId \mid Employee(e) \wedge Department(d) \wedge (\exists s)$$
$$(SecurityOfficer(s) \wedge s.Employee == e \wedge s.Department == d)\}$$

The common dot notation is used to represent values of attributes, as for example in *e.Lastname*, which means the value of attribute *Lastname* in entity *e*. Unary membership predicates like *Employee*(*e*) and *SecurityOfficer*(*s*) are used to indicate that an object belongs to a certain entity or relationship set. Expressions like *s.Employee* are used to denote the role of an entity participating in a relationship. If no role is defined explicitly, as is the case in most relationship sets of our schema, the name of the entity set takes the place of the role. Thus *s.Employee* denotes the entity of type *Employee* that takes part in relationship *s*. We call such expressions *role functions*. The double equality sign '==' means identity of entities or relationships, in contrast to the common equality sign '=', which is only used to express that two values are equal.

The result of the query above is a table consisting of three columns and probably many rows. There will be no two rows in the answer with corresponding values for all columns, because there is only one department/employee combination for every department according to the cardinalities. Despite of this, a query enclosed in { } has a set of rows as result, and a set will never contain two or more identical rows.

2) Lastnames of all employees who are superior to at least one employee with lastname 'Johnson' and forename 'Bill'.

$$\{e1.Lastname \mid Employee(e1) \wedge (\exists e2,h)(Employee(e2) \wedge Hierarchy\ (h) \wedge h.Superior == e1$$
$$\wedge h.Subordinate == e2 \wedge e2.Lastname = \text{'Johnson'} \wedge e2.Forename = \text{'Bill'})\}$$

As *Hierarchy* is a recursive relationship set there must be explicit role declarations for the participating entities, otherwise the cardinalities could not be interpreted. Correspondingly, there are role functions with true role names in the query. *h.Superior* represents the employee participating in relationship *h* who is superior to employee *h.Subordinate*, who also takes part in *h*.

3) The same as before except that now identical rows should also be printed. A query enclosed in [ ] yields a list as result. That means that double rows are allowed.

$$[e1.Lastname \mid Employee(e1) \wedge (\exists e2,h)(Employee(e2) \wedge Hierarchy\ (h) \wedge h.Superior == e$$
$$\wedge h.Subordinate == e2 \wedge e2.Lastname = \text{'Johnson'} \wedge e2.Forename = \text{'Bill'})]$$

4) Number of employees in department 'Marketing'

$$Count[e\ {}^{*} \mid Employee\,(e) \wedge (\exists d,m)(Department(d) \wedge MemberOf\ (m) \wedge$$
$$m.Employee == e \wedge m.Department == d \wedge d.DeptName = \text{'Marketing'})]$$

Count is an aggregate function which takes a list or a set of rows as argument and returns a single value. Other aggregate functions are *Sum*, *Min*, *Max*, and *Avg*. There are also 'normal' functions in ERC which take one ore more single values as arguments. We will use some of them in the following chapter.

3 Modeliing Derivable Data

3.1 Language Extensions to Define Derivable Components

The declaration of derivable schema components in an ER schema consists of two parts:

- The components have to be declared in the schema description language like original components. In the ER-diagram, they may be marked as being derivable, but this is not a must. In the ER diagram of Figure 1, derivable components are drawn in dotted lines, but in the verbal description of Figure 2 no distinction has been made between original an derivable data.
- Data from other schema components have to be assigned to every derivable component. Exactly one assignment is needed for a derivable component. This assignment has two tasks: Firstly, it marks the component as being derivable, secondly, it is the manifestation of the derivation rule.

We use the common symbol ':=' as assignment operator. Assignments to derivable components have the general form

< schema component > := < query >.

By such an expression, the result of the query on the right side is assigned to the derivable schema component mentioned on the left. Dependent on the character of the derivable component, some extensions to ERC are needed.

Assignment to a Subset of an Entity Set

If the derivable entity set is a *subset* of another entity set which has already been defined in the schema, the assignment takes one out of two forms:

< entity set name > := {< entity variable > | < condition > },
< entity set name > := { < entity variable > , < value list > | < condition > }

The first form has to be applied if the derivable entity set has no attributes besides those inherited from its superset. If the derivable entity set has additional attributes (which must be derivable as well) the second form has to be taken.

5) Suppose, a rule in our firm says that if a security officer has not attended a first aid course for a year he has to be trained immediately. We may define an entity set *EmployeeToBeTrained*, which is a subset of *Employee*, to hold all the persons concerned. If there are no additional attributes, we take the first form of assignment:

$$\begin{aligned}&\mathit{EmployeeToBeTrained} := \\ &\{e \mid \mathit{Employee}(e) \wedge \mathit{datediff}(\text{Today}, e.\mathit{DateOfLastFirstAidCourse}) > 365 \\ &\wedge (\exists s)(\mathit{SecurityOfficer}(s) \wedge s.\mathit{Employee} == e)\}\end{aligned}$$

6) In order to express how urgent training is, we could add an attribute *Delay* which is the time between the date of the last course and today. Now we have to take the second form of assignment:

$$\begin{aligned}&\mathit{EmployeeToBeTrained} := \\ &\{e, \mathit{datediff}(\text{Today}, e.\mathit{DateOfLastFirstAidCourse}) - 365 \mid \mathit{Employee}(e) \wedge \\ &\mathit{datediff}(\text{Today}, e.\mathit{DateOfLastFirstAidCourse}) > 365 \\ &\wedge (\exists s)(\mathit{SecurityOfficer}(s) \wedge s.\mathit{Employee} == e)\}\end{aligned}$$

Derivable subsets are only a special form of derivable entity sets, but they are very important in practice. Defining other derivable entity sets is also possible, though a little bit more complicated, because they have to be embedded in the rest of the schema by defining one or more relationship sets in addition.

Assignment to a Relationship Set

There are also two types of assignment, depending on whether the relationship set has (derivable) attributes or not:

< relationship set name > := { < participant list > | < condition > },
< relationship set name > := { < participant list > ; < value list > | < condition > }

Therein < participant list > is a list of the entities which take part in the relationship. The number of participants corresponds to the arity of the relationship set. If an entity set plays more than one role in a relationship set, there is a participant in the list for each of these roles.

7) Suppose our firm wants to care for retired employees, e.g. by inviting them to special meetings. The last department in which the employee worked before his retirement is responsible for this contact. In order to get the pairs of corresponding departments and employees, we may establish a derivable relationship set *DeptRetiree* between *Department* and *Employee*. As we need no additional attributes, we have to take the first of the two types of assignment mentioned above.

$$\mathit{DeptRetiree} =$$

$$\{d, e \mid \mathit{Department}(d) \wedge \mathit{Employee}(e) \wedge e.\mathit{Status} = \text{'retired'} \wedge (\exists m1)(\mathit{MemberOf}(m1) \wedge$$

$$m1.\mathit{Department} == d \wedge m1.\mathit{Employee} == e \wedge \neg(\exists m2)(\mathit{MemberOf}(m2) \wedge$$

$$m2.\mathit{Employee} == e \wedge m2.\mathit{To} > m1.\mathit{To}))\}$$

8) In our next example we refer to a recursive derivable relationship set without additional attributes. Let us assume that every employee of the firm has to get a security instruction by the security officer responsible for his department. To get all pairs of instructors and persons to be instructed, we may define a derivable relationship set *SecurityInstruction* in which *Employee* participates in two roles.

$$\mathit{SecurityInstruction} =$$

$$\{e1, e2 \mid \mathit{Employee}(e1) \wedge \mathit{Employee}(e2) \wedge (\exists s, m)(\mathit{SecurityOfficer}(s) \wedge$$

$$\mathit{MemberOf}(m) \wedge s.\mathit{Employee} == e1 \wedge m.\mathit{Employee} == e2$$

$$\wedge s.\mathit{Department} == m.\mathit{Department} \wedge \neg(e1 == e2))\}$$

Assignment to an Attribute

In the ERM every entity and relationship has only one single value for each of its attributes. Thus the query on the right side of the assignment must not result in more than one value. Depending on whether the attribute belongs to an entity set or a relationship set, we use one out of two forms of expression.

< entity set name >.< attribute name > = < value > | < condition >,
< relationship set name >.< attribute name > = < value > | < condition >.

9) The following assignment provides the data for the derivable attribute *AvgTimeSpent* in entity set *Department*. This attribute is used as an indicator for the amount of fluctuation in a department.

$$\mathit{Department}.\mathit{AvgTimeSpent} =$$

$$\mathit{Avg}[\mathit{datediff}(e.\mathit{To}, e.\mathit{From}) \mid \mathit{MemberOf}(m) \wedge$$

$$m.\mathit{Department} == \mathit{Department} \wedge m.\mathit{To} = \mathit{null})]$$

There is no explicit entity variable necessary for the department. Instead, the name of the entity set itself is used as an implicit variable. When the assignment is executed, *Department* takes the values of every department, one after another. Thus all departments fulfilling the condition are supplied with a value for *AvgTimeSpent*.

3.2 Transitive Closures, Recursive Assignments, and Safety of Assignments

The above examples of derivable data can all be transformed to the relational model and be handled in a relational database system by using the enduser facilities only. There are some rules, however, that must be observed to ensure that assignments be executable. Consider, for example, the following assignment:

$$NotEmployee := \{e \mid \neg Employee(e)\}$$

Obviously, the database system cannot enumerate the members of the derived entity set *NotEmployee* because it is infinite. To avoid such troubles, some rules for *safety* of queries have been formulated in the context of TRC (cf. Ullman 1988, pp. 158). As ERC is based on TRC, these rules may be applied in an adapted version.

There are types of derivable data that cannot be handled by nowadays' relational database systems if we restrict ourselves to enduser facilities. Many problems, for instance, require the calculation of *transitive closures*. ERC allows the definition of transitive closures, but such constructs can only be implemented with additional programming efforts. Let us give an example.

10) *Hierarchy*, which is an original relationship set, contains all pairs of persons such that one of them is directly subordinate to the other. If we want all pairs of persons with one of them *directly or indirectly* subordinate to the other, we may define a derivable recursive relationship set *TotalHierarchy*, which is the transitive closure of *Hierarchy*.

$$\begin{aligned}
&TotalHierarchy := \\
&\{e1, e2 \mid Employee(e1) \wedge Employee(e2) \wedge \\
&((\exists h)(Hierarchy(h) \wedge h.Superior == e1 \wedge h.Subordinate == e2) \vee \\
&(\exists h, t)(Hierarchy(h) \wedge TotalHierarchy(t) \wedge h.Superior == e1 \wedge \\
&h.Subordinate == t.Superior \wedge t.Subordinate == e2))\}
\end{aligned}$$

Not only relationship set *TotalHierarchy* is recursive, but also the assignment used to define it, because it refers to itself in the query on the right. Nevertheless, the assignment is executable and yields a reasonable result, because it contains a reference to the original relationship set *Hierarchy* in the third line. We may consider this as the *basis case*. If we want to judge whether there is a basis case for an assignment or not it is often not sufficient to look at only one assignment, because there might be an indirect recursion. The following two assignments, for instance, contain a vicious circle, although there is no direct recursion in any of them:

$$ForeignDepartment := \{d \mid DepartmentAbroad(d)\},$$
$$DepartmentAbroad := \{d \mid ForeignDepartment(d)\}$$

If we use assignments which are directly or indirectly recursive we have to care for a basis case. Again, this is a problem of safety. The interested reader may get further information on this subject in the literature on deductive database systems, e.g. Ullman (1988), Ceri et al. (1990), and Gardarin et al. (1989).

4 Two Classes of Derivable Data

Now we want to introduce a classification of derivable data that has great significance for the treatment of those data in conceptual modelling (cf. Rauh et al. 1993 for details). The basic criterion used to classify the data is the source of their meaning to the user. Let us go back to the example, where a relationship set *SecurityInstruction* was assigned the pairs of instructors and persons to be instructed. The basis for this assignment was a rule saying that the employees working in the departments for which a certain security officer is responsible, should be instructed by this officer. Please note that there has been information added to the system by this rule, and this additional information has been expressed in the name of the derivable component.

Now let us change the assumptions to illustrate another kind of derivable data. Assume now that there is no rule concerning security instructions. Nevertheless, we could define a relationship set consisting of the same relationships as in *SecurityInstruction*. In the assignment for this relationship set we would use the same query as for *SecurityInstruction*. But now 'SecurityInstruction' would not be an appropriate name for it. Without a rule, the schema component on the left side of an assignment has only the meaning that is expressed by the query on the right. As we use a descriptive language we can recognize this meaning at a glance. Considering the query, an appropriate name would now be *SecurityOfficerAndPersonInADepartmentForWhichHeIsResponsible*. This rather clumsy name emphasizes the difficulties that we often have when we look for a name that is both expressive and short. But even if we chose the name *X* for our relationship set, there would be no difficulties for the user to get its meaning if only he knew the right side of the assignment.

We call schema components like *SecurityInstruction*, which have an additional meaning besides that expressed by the query, *s-derivable*. The *s* stands for *special*, as there is a special derivation rule which provides the additional meaning. If there is no such rule, as is the case in the second version of our example, a derivable schema component gets its meaning only from the meaning of the data referred to in the query and from the meaning of the language operations applied. Thus the query language is the basis for all derivations of this kind. In other words, it provides a set of meta derivation rules for them. Therefore we call such components *derivable by meta derivation rules*, or simply *m-derivable*.

Classifying data as being m-derivable or s-derivable is helpful when we have to decide if they should be included in the conceptual schema. As m-derivable data do not add any information to the schema they should be omitted. Every user capable of the database language may generate them just as he likes. In contrast, s-derivable components represent knowledge that is not incorporated in the original data and the operations of the database language. If we include them in the conceptual schema the resulting information system will have deductive capabilities, and these capabilities will be visible to the modellers and the users of the schema from the beginning.

There is one condition, however, which s-derivable data in the conceptual schema should meet. As this schema is a long-term plan for the enterprise's data, it should only contain informations that will be of interest in the long run.

5 Advantages and Limits of the Approach

Extending the ERM to include derivable schema components has several *advantages*:

1) Modellers can stay within in the ERM which is a very popular and illustrative language.

2) They get an additional view onto the system. If a deductive system is modelled in logic alone, derivable data are represented in the derivation rules only. Integrated in the ERM, the relationships of the data with other data are visible. This feature is particularly useful if the schema contains a great number of original entity and relationship sets.

3) The extensions provide a basis for a sound treatment of derivable data in the ERM. Up to now, there were no means to handle such data in the ERM. Nevertheless, many modellers included derivable data in their schemas, treating them like original data. But if derivable data are not marked as being derivable the integrity of the database is put at risk.

4) As far as only nonrecursive assignments are used, the conceptual model can easily be transformed into a relational database scheme. This may be done automatically.

There are also some *limits* that have to be observed.

1) If recursive assignments are used, additional programming efforts have to be made to supply the data.

2) In systems with a lot of rules, say some thousands, the clarity and expressiveness, which is normally an advantage of ER schemas, might no longer be existent. Also, the benefits of the approach are moderate if the system consists of very few entity sets, whereas the derivation rules result in a great number of relationship sets.

3) As ERC is based on TRC and TRC is, on the other hand, based on first order predicate logic, ERC is restricted to the same applications of reasoning as first order logic is.

References

Ceri, S., Gottlob, G., Tanca, L., (1990), "Logic Programming and Databases", Springer-Verlag, Berlin.

Chen, P.P.S., (1976), The Entity-Relationship model: Toward a unified view of data. *ACM TODS* 1, No. 1, March 1976.

Elmasri, R. and Wiederhold, G., (1981), GORDAS: A formal, high-level query language for the Entity-Relationship model, *in*: "Entity-Relationship Approach to Information Modelling and Analysis", P.P. Chen, ed., Elsevier.

Gardarin, G. and Valduriez, P., (1989), "Relational Databases and Knowledge Bases", Addison-Wesley, Reading, Mass.

Gogolla, M. and Hohenstein, U., (1991), Towards a semantic view of an extended Entity-Relationship model, ACM TODS, Vol 16, No. 2, Sep 1991, pp. 269-316.

Hohenstein, U., (1990), Automatic transformation of an Entity-Relationship query language into SQL, *in*: "Proceedings of the 8th International Conference on Entity- Relationship Approach", F. Lochovsky, ed., North-Holland, Amsterdam.

Lagrange, J.-P., (1990), A knowledge-based system and an ER query language for accessing relational databases, *in*: "Proceedings of the 9th International Conference on Entity-Relationship Approach", H. Kangassalo, ed., Lausanne.

Lloyd, J.W., (1987), "Foundations of Logic Programming", 2nd edition, Springer-Verlag, Berlin.

Markowitz, H.M., Malhotra, A., Pazel, D.P., (1981), The ER and EAS formalisms for system modeling, and the EAS-E language, *in*: "Entity-Relationship Approach to Information Modeling and Analysis", P.P. Chen, ed., Elsevier.

Parent, C., Rolin, H., Yetongnon, K., Spaccapietra, S., (1990), An ER calculus for the Entity-Relationship Complex model, in: "Proceedings of the 8th International Conference on Entity-Relationship Approach", F. Lochovsky, ed., North-Holland, Amsterdam.

Rauh, O., (1992), Some rules for handling derivable data in conceptual data modeling, *in*: "Database and Expert Systems Applications", A. M. Tjoa and I. Ramos, eds., Springer-Verlag, Wien and New York.

Rauh, O. and Stickel, E., (1993), Searching for compositions in ER schemes, *in*: "Proceedings of the 12th International Conference on Entity-Relationship Approach", R. Elmasri and V. Kouramajian, eds., E/R Institute.

Rauh, O. and Stickel, E., (1994), "Qualitätssicherung in der konzeptionellen Datenmodellierung", to appear.

Rich, E. and Knight, K., (1991), "Artificial Intelligence", 2nd edition, McGraw-Hill, New York.

Ullman, J.D., (1988), "Principles of Database and Knowledge-Base Systems", Vol. I, Computer Science Press, Rockville, Maryland.

Zaniolo, C., (1982), The database language GEM, *in*: "Proceedings of the 1982 ACM-SIGMOD Conference on Management of Data", San Jose, California, May 1982.

MODELLING MULTI-PARTY ACTIVITY IN THE IS PROCESS FROM A LANGUAGE/ACTION PERSPECTIVE

Ian R McChesney

Department of Information Systems, University of Ulster, Newtownabbey, Co. Antrim,
Northern Ireland, UK, BT37 0QB

Abstract

Multi-party activity pervades the IS process, yet remains a problematical phenomenon, lacking sufficiently rigorous formalisms for its detailed modelling and automated support. In this paper, we adopt a language/action perspective on multi-party activity as the basis for developing this aspect of IS process modelling. We consider the nature of multi-party activity in the IS process, and illustrate how it may be modelled using an existing speech-act-based modelling technique. We discuss the adequacy of this technique for IS process modelling, and in this way identify future directions for investigation and development in this area.

1. INTRODUCTION

The need to conduct the software development process in a structured and disciplined manner has been recognized from the early years of the software industry (Royce, 1970). While many methodologies for software development have been proposed (for example, Jackson, 1983, CCTA, 1990), only in recent years has the distinct concept of the *software process* emerged as the subject of widespread research and investigation (Dowson, 1991, IEEE, 1993; Madhavji and Schafer, 1991; Ince and Tully, 1993). A major characteristic of such work is the *modelling* of the software process with a view to its improved understanding, automation and control.

Given the central position of software in modern information systems (IS), it is clear that advances in the software process have considerable relevance to the *IS process*, (where the IS process may be informally defined as that collection of policies, procedures and steps undertaken in the transformation of an expressed need for an IS into an IS to meet that need). However, certain characteristics of IS development distinguish it from software systems development. These differences relate primarily to the early stages of an IS project (such as strategic planning of information systems and requirements definition) which tend to be less structured and mechanistic, and more human intensive, than later stages (Tamai, 1993). The scope of the IS process includes all of those activities associated with the software process, and in addition those lifecycle activities which may not be generating software artefacts, but are defining the environment and terms of reference for the underlying software process. These differences in the IS processes do not require a radically different approach to modelling and support, but rather a shift in emphasis.

In this paper, we consider the modelling of the IS process with particular emphasis on the human-intensive process of *multi-party activity*, (i.e. those tasks in IS development requiring the coordinated action of two or more individuals to achieve a common goal). Multi-party activity pervades the IS process (e.g. quality reviews, project planning, brainstorming, requirements definition, debugging, acceptance testing, project control and monitoring). Such coordinated activity depends upon (explicit and/or tacit) communication between participants in order to, for example, establish a course of action or report on the status of individual tasks. Adequate support for multi-party activity requires firstly, a clear understanding of the communication and coordination mechanisms between individuals, and secondly, the availability of suitable formalisms for modelling such mechanisms. In the following sections we investigate one approach to modelling such activity.

The remainder of the paper is organized as follows. The next section presents a brief review of related work in the field of software process modelling. In section three, we consider the nature of multi-party activity in the IS process, and its relationship to other types of activity. Section four introduces the notion of a language/action perspective on the IS process. In section five we use an existing approach, based on the notion of 'speech acts', to model such activity, and on this basis consider the desirable features of a formalism for modelling multi-party activity in the IS process. Section six summarizes the major points of the paper.

2. RELATED WORK

Software process modelling is concerned with modelling the agents, activities and artefacts in the software development process. A large number of approaches have been proposed and these are usefully considered in terms of their underlying representation style, for example:

- process programming (Sutton et al, 1990, Ramanathan and Sarkar, 1988)
- AI/knowledge-based approaches (Huff and Lesser, 1988, Kaiser et al, 1988)
- non-deterministic finite-state automaton (Deiters and Gruhn, 1991)
- functional programming approach (Katayama, 1989)
- object-oriented approaches (Lonchamp et al, 1991, Belkhatir et al, 1993)

The majority of such work has generally focused on software engineering related technical and management issues, only in recent years has the role of communication and coordinated activity in the software process been considered in detail (e.g. , Kaplan et al, 1992; Barghouti, 1992; Cain and Coplien, 1993; Finkelstein et al, 1992).

3. THE NATURE OF MULTI-PARTY ACTIVITY IN THE IS PROCESS

Gruhn (1992) distinguishes three types of activity in the software process:

- *automatic activities*: activities carried out without human-interaction, such as compilation of a module

- *individual activities*: activities that require the involvement of one participant, such as editing a program.
- social interaction (or *multi-party*) activities: activities that require interaction between two or more participants, such as reviewing a design specification.

Automatic and individual activities in the IS process are relatively well understood and enjoy substantial automated support in the form of product-centred and process-centred computer-aided software engineering tools. However, multi-party activity remains problematical and lacks the modelling formalisms necessary for its sophisticated support.

The human-intensive nature of multi-party activity leads to its inherent uncertainty and complexity. Even though such activity may be carefully planned and adequately resourced, it is prone to uncertain outcomes, ambiguity, inconsistency and incompleteness, and even where multi-party activity progresses in an optimum manner, it remains a resource intensive process, typically consisting of tasks which are inter-leaved and dependent upon other (multi-party and individual) activities, and often concerned with those issues which are critical to the overall success of the project (e.g. requirements specification, change control or project monitoring).

4. A LANGUAGE/ACTION PERSPECTIVE ON THE IS PROCESS

4.1 Speech Acts - the foundation for a language/action perspective

One approach for modelling multi-party activity is to focus on the role of *language* and *conversation* in cooperative activity (Winograd and Flores, 1986). In the field of CSCW (computer-supported cooperative work) in general, and software process modelling in particular, a number of paradigms within this school of thought have been applied, e.g.:

- *Language action perspective (Speech Acts)* (Searle, 1969; Winograd and Flores, 1986): used in projects/tools such as The Coordinator (Winograd, 1988),Conversation Builder (Kaplan et al, 1992) and Alf (Lonchamp, 1992).
- *Dialogue logics* (e.g. Hamblin, 1987): used in tools for multi-party specification (Finkelstein and Fuks, 1989).
- *Conversation Analysis* (Wooffitt, 1990): applied to the field of human-computer interaction.
- *Principles and maxims of conversation* (Grice, 1975): applied to the context of conversation repair in human-computer interaction (Good, 1990).

It is not clear that any one of the above paradigms is the most appropriate for modelling multi-party activity in the IS process. However, we have identified the language/action paradigm as an approach which has been sufficiently developed to permit its use in investigating such activity. The language/action perspective holds that multi-party activity in organizations is achieved through *language* (Winograd and Flores, 1986; Winograd, 1988), (the more traditional view being that multi-party activity is achieved through people processing information and communicating decisions). Winograd and Flores (1986) have built upon this position and developed its relevance to the design of computer-based systems. The basis for this perspective is drawn from work in the philosophy of language concerning *Speech Acts* (Austin, 1962; Searle, 1969). Speech acts may be considered as 'performative utterances', i.e. utterances in which the speaker is said to be performing some action, (e.g. declaring a conference open, ordering someone to do something, telling somebody how you feel) Searle (1975) proposes that such performative utterances can be classified according to the following scheme

- *assertives* utterances in which the speaker presents a proposition as representing an actual state of affairs (e g reporting an error in a design specification)
- *commissives* utterances in which the speaker commits him/herself to carrying out some future course of action (e g promising to modify a program)
- *directives* utterances in which the speaker attempts to get the hearer to carry out some future course of action (e g questioning or issuing a command)
- *declarations* utterances in which the speaker brings about the state of affairs represented by the propositional content of the utterance, solely by virtue of his/her successful performance of the utterance (e g pronouncing a employee 'fired')
- *expressives* utterances in which the speaker expresses some psychological attitude about a state of affairs (e g apologising or praising)

The language/action perspective is concerned with the study of multi-party activity in terms of the speech acts of the participants In this approach, speech acts are considered as the fundamental unit of discourse, and any discourse or discourse segment is composed of a series of speech acts, sequenced according to certain patterns or rules of discourse (Searle and Vanderveken, 1985, Taylor and Cameron, 1987, Bowers and Churcher, 1988)

Sacks et al (1974) note that "speech exchange systems" may be considered as a spectrum of exchanges types At one end of the spectrum lie conversations, which are said to be *locally structured* (often only the next turn being determinate at any point in time), at the other extreme we have cooperative activity where every turn and order is predetermined (as in ceremonies or rituals) Bowers and Churcher (1988) consider examples of the latter case to be *globally structured* discourse In many organizations effective coordination of multi-party activity is achieved through such global management of communicative action (for example, procedures for writing a report, purchasing materials, modifying a product design) The sequencing of speech acts according to global structuring rules is of particular interest from an IS process point of view, as we shall illustrate in the following section

4.2 Speech acts in the IS process

Scenario: Quality Reviews

Quality review processes, such as Structured Walkthroughs or Software Inspections, are group activities conducted according to specific procedures, involving social interaction and information exchange For example, consider the scenario of a Software Inspection (Fagan, 1976) where a design engineer submits a design modification for review by a review team under the direction of a review moderator (Kellner et al, 1991) Analysing such interaction from a language/action perspective, the speech acts identified might include the following

- design engineer *describes* the design modifications to the review team (*assertive*)
- moderator *requests* the design engineer to make minor changes (*directive*)
- design engineer *agrees* to making the minor changes within given timescale (*commissive*)

In a Software Inspection, such speech acts are conducted according to global structuring rules; for example, the pointing out of errors and their confirmation is repeated for every defect identified in the design, the conclusion of the review meeting is followed by one of three speech acts (announce approval or request minor changes or request major changes). Within this global framework, locally structured conversations may also take place, such as the process of clarifying errors during the review meeting, or negotiating the commitment to make the change by a particular date.

5. MODELLING SPEECH ACTS IN MULTI-PARTY ACTIVITY

5.1 SAMPO (Speech-Act-based Modelling aPprOach)

SAMPO (Auramaki et al. 1988, Auramaki et al. 1992) is a collection of techniques devised to model offices as systems of communicative action. Through such communicative action (or "multi-party activity"), participants create, modify and nullify commitments which bind their future and current behaviours. In SAMPO, multi-party activity is modelled at the discourse level in terms of speech acts between participants. SAMPO proposes two graphical description techniques for discourse analysis, supplemented with tabular representations of discourse entities.

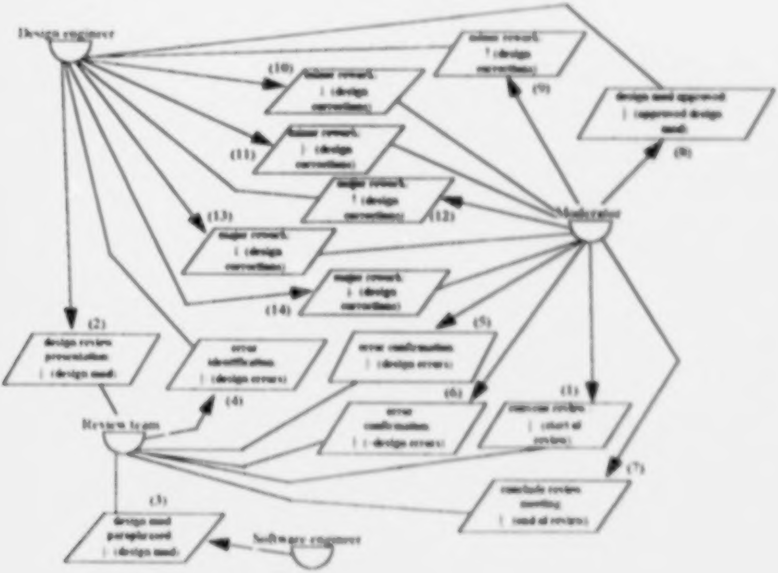

Design engineer
Moderator
Review team
Software engineer
(1)
(2)
design review presentation
(3)
(4)
error identification
(5)
error confirmation
(6)
error confirmation
(7)
conclude review meeting
(8)
design used approved
(9)
(10)
minor remark
(11)
minor remark
(12)
major remark
(13)
major remark
(14)
major remark

Figure 1 Discourse graph for software inspection

A *discourse graph* delineates discourse objects and their relationships (figure 1). It seeks to model the global or 'institutionalised' structuring of the discourse, defining the necessary and sufficient conversation possibilities for each participant. A *conversation graph* seeks to describe patterns of speech acts and their dynamic dependencies (figure 2). Conversations graphs do not assume a fixed discourse structure, rather they provide the constructs for describing domain specific communication patterns. Using these two

EST COPY AVAILABLE

techniques, SAMPO may be used for discourse analysis, in particular the assessment of the discourse under consideration with respect to discourse coherency, completeness, ambiguity, and the coordination of commitments made in the discourse. For example, discourse completeness is investigated using the petri-net properties of a conversation graph through construction of its reachability tree, identifying the possible terminating discourse paths.

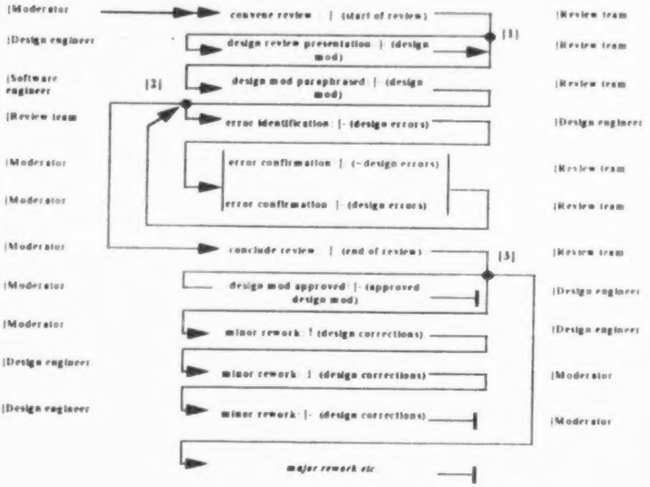

|Moderator
convene review : | (start of review)
|Review team
|Design engineer
design review presentation : | (design mod)
[1]
|Review team
|Software engineer
[2]
design mod paraphrased : |- (design mod)
|Review team
|Review team
error identification : |- (design errors)
|Design engineer
|Moderator
error confirmation : | (~design errors)
|Review team
|Moderator
error confirmation : |- (design errors)
|Review team
|Moderator
conclude review : | (end of review)
[3]
|Review team
|Moderator
design mod approved : |- (approved design mod)
|Design engineer
|Moderator
minor rework : ! (design corrections)
|Design engineer
|Design engineer
minor rework : | (design corrections)
|Moderator
|Design engineer
minor rework : |- (design corrections)
|Moderator
major rework etc

Figure 2. Conversation graph for software inspection

5.2 Modelling the IS process using SAMPO

The author has informally investigated the relevance of SAMPO for IS process modelling through applying it to a small number of IS development activities, such as quality review processes (e.g. figures 1 and 2), project scheduling and control, and brainstorming. The preliminary conclusions arising from this are that a SAMPO-like approach is useful for making visible the dialogue of multi-party activity within the IS process, and its underlying features provide a basis for developing a speech-act based approach supporting:

- the visualisation of multi-party activity.
- the modelling of high-level discourse structure, identifying cohesive discourse segments for further analysis.
- qualitative identification of focal points in multi-party activity in IS process (along the lines of the CRC-card approach (Cain and Coplien, 1993)).
- the specification of global structuring of IS multi-party activity.
- identification of omissions in discourse with respect to coherency and completeness of multi-party activity.

BEST COPY AVAILABLE

However, in its current form, SAMPO is considered to support only a rather loose and passive form of IS process modelling, and will require further development in some specific areas to enhance its suitability for this task. These proposed developments are discussed in the following sections.

Coordination of commitments

A key objective of modelling processes from a language/action perspective is to study the *coordination* of tasks and commitments between participants. In SAMPO this is undertaken through constructing a network in which is shown the relationships between speech acts and the evolution of commitments. However, the documented approach does not make clear the precise meaning of "relationships" between speech acts, nor the impact of a commitment change to the 'state of a speech act'. We propose that an underlying problem is that commitments in SAMPO are not considered in terms of the physical tasks (or "instrumental acts" in SAMPO terminology) to which they relate. While commitments are made and modified through speech acts, it is a commitment to undertake a particular task, and hence the relationship between the commitment and the task which is significant.

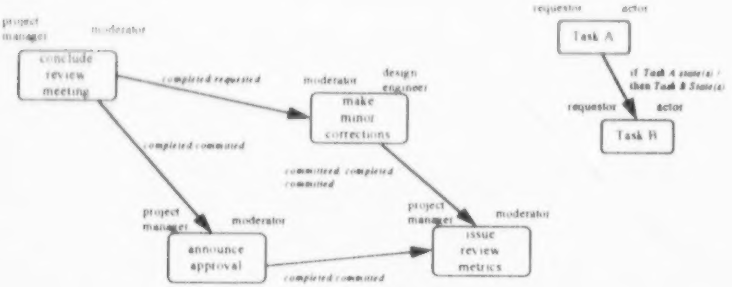

project manager
moderator
conclude review meeting
completed requested
moderator
design engineer
make minor corrections
completed committed
committed completed committed
project manager
moderator
announce approval
project manager
moderator
issue review metrics
completed committed
requestor
actor
Task A
if Task A state(s) / then Task B State(s)
requestor
actor
Task B

Figure 3 Commitment network of multi-party tasks

Further work, therefore, would need to investigate the precise relationship between a commitment to some task (made in a speech act), the state of the task in question (e.g. 'ready', 'committed', 'pending', 'cancelled', 'completed'), and the evolution of its state as the commitment is followed through to completion. Physical tasks and their inter-relationships may be modelled as a network of tasks, indicating for each task its requester (speaker) and actor (hearer), and its dependency relationship to other tasks (figure 3). Within such a network, the dependency relationships *between tasks* may be modelled in terms state-pairs (i.e. if task A is in state 'requested', task B must also be 'requested'; when task A has been 'completed', task B must be 'committed').

Local and global structuring

The modelling techniques in SAMPO do not address the local structuring of dialogue. Their concern is with the high-level organization of speech acts between participants and their global structuring. However, the local structuring of dialogue does have relevance when we are constructing process models as the basis for *automated support* in an IS development environment. The importance of the local structuring of conversations has been addressed in the COSMOS project (Bowers et al, 1988) where the objective was to provide a method of defining conversation structures as the basis for sophisticated and customised computer-mediated communication (CMC) systems. This work has considered the relevance of locally structured conversation features to dialogue support in CMC systems. For example, the role of asynchronous message transmission, adjacency pairs and turn-taking in CMC.

We propose that such issues are highly relevant for process-based IS development environments. In such cases, SAMPO-type models may be used to specify global structuring of communicative action, with identified departure points where local conversations may be undertaken through the medium of the development environment. This is particularly relevant where we have computer-supported, distributed IS development.

Methodology for dialogue analysis in the IS process

SAMPO does not address how the content and structure of communicative action is captured in the first instance. A third area for further investigation, therefore, is to *empirically identify* the format and structure of discourse for multi-party activity in the IS process. This will need to draw upon current methodology in conversational analysis (Wooton, 1989) and consider its relevance and implementation in the context of the IS process.

6. CONCLUDING REMARKS

Our goal in modelling the IS process from a language/action perspective is to construct process models which accurately describe and support communicative action within the IS process. To this end, we have focused on the communicative process in terms of speech acts, the relationships between speech acts, the associated instrumental tasks and the coordination of commitments between participants within a project. Techniques such as those found in SAMPO are examples of how such a perspective might be modelled, and have been presented in this paper in order to stimulate further thinking on their relevance to the IS process and how they might be further developed.

It is proposed that a number of benefits would arise from a language/action perspective on IS process modelling, for example:

- the ability to analyse multi-party activity in a formalised and structured manner, and assess the completeness and coherency of IS processes.
- the ability to prescribe processes with a view to their automated support, in particular support for multi-party activity in a computer-mediated distributed environment (e.g. such support might involve restriction of interaction to permitted activities, and/or checks for deviations from recommended procedures).
- the monitoring of project status taking into consideration the impact and frequency of communicative action (Schafer, 1993) (e.g. in determining: all tasks currently 'requested', all tasks 'committed', all tasks waiting on a commitment to be fulfilled, all tasks committed to by a particular individual, all tasks requested by a particular individual, which tasks are 'ready' pending other requests or commitments to be made).

The achievement of such benefits would result in an improved theoretical understanding of the IS process, and also extend the management of IS projects to take explicit account of cooperative activity, providing the basis for more informed and effective project management.

7. REFERENCES

Auramaki, E., Lehtinen, E., and Lyytinen, K., (1988). A speech-act-based office modelling approach. *ACM Trans. Office Info. Sys.*, Vol 6 No 2 (April 1988), pp. 126-152.

Auramaki, E., Hirschheim, R., and Lyytinen, K., (1992). Modelling offices through discourse analysis: the SAMPO approach. *Computer J.*, Vol 35 No 4, pp. 342-352.

Austin, J., (1962). "How to do things with words", Harvard University Press, Cambridge, USA.

Barghouti, N., S., (1992). Cooperation in the MARVEL process-centred SDE. *ACM SIGSOFT Soft. Eng. Notes*, Vol 17 No 5 (December 1992) pp 21-31.

Belkhatir, N., Estublier, J., and Melo, W., L., (1993). Software process model and work space control in the Adele system. *in* "Proc. 2nd Int. Conf. Software Process, Berlin, Germany, (February 25-26, 1993)" pp 2-11.

Bowers, J., and Churcher, J., (1988). Local and global structuring of Computer Mediated Communication: developing linguistic perspectives on CSCW in COSMOS. *in* "Proc. Conf. on Computer-Supported Cooperative Work, September 26-29, 1988, Portland, Oregon, USA", pp. 125-139.

Bowers, J., Churcher, J., and Roberts, T., (1988). Structuring computer-mediated communication in COSMOS. *in* "EUTECO'88, Research into Networks and Distributed Applications", R. Speth, ed., Elsevier Science Publishers.

Cain, B., G., and Coplien, J., O., (1993). A role-based empirical process modelling environment. *in* "Proc. 2nd Int. Conf. Software Process, Berlin, Germany, (February 25-26, 1993)", pp 125-133.

CCTA (Central Computer and Telecommunications Agency), (1990). "SSADM Version 4 Reference Manual", NCC Blackwell, Oxford, UK (1990).

Deiters, W., and Gruhn, V., (1991). Software process analysis based on funsoft nets. *Systems Analysis Modelling Simulation*, Vol 8 No 4-5, pp 315-325.

Dowson, M., (ed), (1991). "Proc. 1st Int. Conf. Software Process, October 21-26, 1991, Redondo Beach, California, USA", IEEE Computer Society Press, Los Alamitos, USA.

Fagan, M., E., (1976). Design and code inspections to reduce errors in program development. *IBM Syst. J.*, Vol 15 No 3, pp. 182-211.

Finkelstein, A., and Fuks, H., (1989). Multi-party specification. *ACM SIGSOFT Soft. Eng. Notes*, Vol 14 No 3 (May 1989), pp. 185-195.

Finkelstein, A., Kramer, J., Nuseibeh, B., Finkelstein, L., and Goedicke, M., (1992). Viewpoints: a framework for integrating multiple perspectives in system development. *Int. J. Soft. Eng. and Knowl. Eng.* Vol 2 No 1 (1992) pp 31-57.

Good, D., (1990). Repair and cooperation in conversation. *in* "Computers and Conversation", Academic Press, London, UK, pp. 133-150.

Grice, H., P., (1975). Logic and conversation. *in* "Syntax and Semantics: Speech Acts", P. Cole and J. L. Morgan, eds., Academic Press, New York, USA.

Gruhn, V., (1992). Software processes are social processes. *in* "Proc. Fifth Int. Workshop on Computer-Aided Software Engineering, Montreal, Quebec, Canada, July 6-10, 1992", pp. 196-201.

Hamblin, C., (1987). "Imperatives", Blackwell, Oxford.

Huff, K., E., and Lesser, V., R., (1988). A Plan-based intelligent assistant that supports the software development process. *ACM SIGSOFT Soft. Eng. Notes*, Vol 13 No 5, (November 1988) pp 97-106.

IEEE Computer Society, (1993). "Proc. 2nd Int. Conf. Software Process, Berlin, Germany, February 25-26", IEEE Computer Society Press, Los Alamitos, USA.

Ince, D., and Tully, C., (eds), (1993). "Inf. Soft. Technol. - Special Issue on Software Process Modelling in Practice", Vol 35 No 6/7 June/July 1993.

Jackson, M., A., (1983). "System Development", Prentice-Hall, Hemel Hempstead, UK.

Kaiser, G., E., Feiler, P., H., and Popovich, S., S., (1988). Intelligent assistance for software development and maintenance. *IEEE Software*, (May 1988) pp 40-49.

Kaplan, S., M., Tolone, W., J., Carroll, A., M., Bogia, D., P., and Bignoli, C., (1992), Supporting collaborative software development with Conversation Builder, *ACM SIGSOFT Soft. Eng. Notes*, Vol 17 No 5 (December 1992) pp 11 20.

Katayama, T., (1989), A hierarchical anf functional software process description and its enaction, *in*: "Proc. 11th Int. Conf. Software Engineering, Pittsburgh, PA, USA (May 15-18, 1989)", pp 343 352.

Kellner, M., I., Feiler, P., H., Finkelstein, A., Katayama, T., Osterweil, L., O., Penedo, M., H., and Rombach, H., D., (1991), ISPW-6 Software Process Example, *in*: "Proc. 1st Int. Conf. Software Process, October 21-26, 1991, Redondo Beach, California, USA", IEEE Computer Society Press, Los Alamitos, USA, pp. 176 186.

Lonchamp, J., (1992), Supporting social interaction activities of software processes, *in*: "Proc. 2nd European Workshop on Software Process Technology, EWSPT'92, Trondheim, Norway", (September 7-8, 1992)", pp. 34 54.

Lonchamp, J., Benali, K., Derniame, J., C., and Godart, C., (1991), Towards assisted software engineering environments, *Inf. Soft. Technol.* Vol 33 No 8 (October 1991) pp 581 593.

Madhavji, N., H., and Schafer, W., (eds), (1991), "Soft. Eng. J. - Special issue on software process and its support", Vol 6 No 5, September 1991.

Ramanathan, J., and Sarkar, S., Providing customized assistance for software lifecycle approaches, *IEEE Trans. Soft. Eng.* Vol 14 No 6, June 1988, pp 749 757.

Royce, W., W., (1970) Managing the Development of Large Software Systems, *in*: "Proc. of IEEE WESCON, 1970, pp. 1 9.

Sacks, H., Schegloff, E., A., and Jefferson, G., (1974), A simplest systematics for the organization of turn-taking for conversation, *Language*, Vol 50, pp. 696-735.

Schafer, W., (1993), Introduction and summary of workshop, *in*: "Proc. 8th Int. Software Process Workshop, March 2-5, 1993, Wadern, Germany", IEEE Computer Scoiety Press, Los Alamitos, USA.

Searle, J., R., (1969), "Speech Acts", Cambridge University Press, Cambridge, UK.

Searle, J., R., (1975), A taxonomy of illocutionary acts, *in*: "Language, Mind and Knowledge", K. Gunderson, ed., University of Minnesota Press, Minneapolis, USA, pp. 344 369.

Searle, J., R., and Vanderveken, D., (1985), "Foundations of Illocutionary Logic", Cambridge University Press, Cambridge, UK.

Sutton, S., M., Jr., Heimbigner, D., and Osterweil, L., J., (1990), Language constructs for managing change in process-centred environments, *in*: "Proc. 4th Symp. Practical Software Development Environments, Irvine, California, (1990)".

Tamai, T., (1993), Current practices in software process for system planning and requirements analysis, *Inf. Soft. Technol.*, Vol 35 No 6/7 June/July 1993, pp. 339 344.

Taylor, T., J., and Cameron, D., (1987), "Analysing Conversation: Rules and Units in the Structure of Talk", Pergamon Press, Oxford, UK.

Winograd, T., (1988), A language/action perspective on the design of cooperative work, *Human-Comp. Interaction*, Vol 3, (1987-1988), pp. 3 30.

Winograd, T., and Flores, F., (1986), "Undersdtanding Computer and Cognition: a New Foundation for Design", Ablex, Norwood, USA.

Wooffitt, R., (1990), On the analysis of interaction - an introduction to conversation analysis, *in*: "Computers and Conversation", Academic Press, London, UK, pp. 7 38.

Wooton, A., J., (1989), Remarks on the methodology of conversation analysis, *in*: "Conversation", Roger, D., and Bull, P., eds., Multilingual MAtters, Clevedon.

Information System Analysis:
A Formal Model for the Specification of Behavioural and Structural Class Properties

Giovanni Rumolo[1], Maurizio Lenzerini[1]

[1]Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza"
Via Salaria 113, I-00198 Roma, Italy. e-mail: `[rumolo,lenzerini]@.dis.uniroma1.it`

Abstract.

This paper presents a new, logic based, model and language for system requirements analysis. The model is able to specify dynamic and static properties of classes of individuals, as well as inclusions and relationships between classes. The model can describe indefinite information, negative properties and sufficient and necessary conditions for classes. We are able to specify processes in a decompositional and compositional way. The operators overloading, the use of the conditional selection, functional composition and application give the language the right expressive power for the functional specification. The taxonomic capacity together with process overloading, characterize the model like a tool for object oriented analysis. Furthermore we promote the bottom up construction of the scheme opening the way to the specification reuse. The most important feature of the language is its deductive capacity. We are able to check consistency for specification schemes, and then we can derive properties from the schemes. Our model comes up with the idea to achieve two goals. One: it is better to give necessary and sufficient tools for the engineers daily work than a set of superfluous, sometimes undecidable, but wonderful tools. The second goal is to have an easy to use and easy to understand model in order to obtain a rapid diffusion in the industrial environment.

1 Introduction.

In the area of software engineering and information system analysis there exists a wide debate regarding models and languages for system specification. We can find different approaches and we can see the contrast between a pragmatic (industrial) point of view against the formal (theoretical founded) approach. We believe that the two points of view must and can meet in an integrated approach that does not sacrifice the feasibility of industrial projects against the necessity of theoretical well founded models. This paper presents a new, logic based, model and language for system requirements analysis.The model is able to specify dynamic and static properties of classes of individuals, as well as inclusions and relationships between classes.The model can describe indefinite informations, negative property and sufficient and necessary condition for objects to belong to classes. This is made in the terminological languages style used in knowledge representation (see [Sch91]). But our model is close to the semantic modeling approach as well (see [HK87]).The dynamic aspect of the system is specified by the processes description activity. We are able to specify processes in both decompositional and compositional way. The operators overloading, the use of conditional selection, functional composition and application give the language the right expressive power for functional specification. The taxonomic capacity together with processes overloading characterize the model like a tool for object oriented analysis.

The most important feature of the language is its deductive capacity obtained through the correspondence with propositional dynamic logic (PDL).This kind of modal logic has a nice property: the logic is decidable. So we are able to check consistency for specification schemes, and then we can derive properties from the schemes too. The complexity of the satisfaction problem for the PDL is EXPTIME and this confirms the fact that the requirements analysis is a reasoning expert task. The correspondence with a decidable, consistent and complete propositional logic, grant that we have a way to automate the reasoning process over the specification schemes. From our study we can deduce that too many proposals in the field of requirements analysis don't have this fundamental property. In the last years the necessity of an anticipate requirements verification is clearly emerged. In information systems development process the requirements phase is critical to obtain the success of the project. The necessity and the possibility of a formal method to check the products of the requirements analysis phase is not only a interesting field of research investigation, but also a first class industrial productivity goal.

The paper is organized as follows: in Section 2, we present a short survey of the literature in the field. In Section 3, we describe how to use the language. In Section 4, we define the model and we give the syntax and the semantics of the language. In Section 5, we show our technique for reasoning about a scheme in order to check consistency of definitions and to compute logical consequences. In Section 6, we discuss some limitations of the model and some relevant questions about the CASE systems.

2 Other related works: a short survey.

The study of other related works in the field bring us to the individuation of five different approaches to the requirements analysis problem. Each of these is presented by the authors like a world apart, with a different lexicon and a different history. The worlds (and then the research areas), don't try to communicate with each other but indeed, they try to be considered as the only global problem solution. We believe that because the problem is common, we need a shared vision of the ontological aspects of the requirements analysis. We haven't found any effort in this direction and we think that this work is necessary (see [Rum93]). We have named the following five worlds as: Semantic Modeling (SM), Object Oriented Analysis (OOA), Knowledge Based Modeling (KM), Deductive Conceptual Modeling (DCM) and Linguistic Based Modeling (LM).

In the SM area[HK87], historically the first, we have included proposals that are closest to Entity-Relationship (E-R) (see [Che76]) and Data Flow Diagram (DFD) methodologies. We have analyzed proposals that deal with structural analysis problem like IFO[AH87], but also integrated models like [EGH$^+$92], or [Kun89]. All the authors are influenced by the relational databases theory and thus their models are weak on the system functions specification. Usually the union of the static scheme to the dynamic part results in a problem of integrate scheme checking. In our model we have avoided the problem using an integrate framework for both behavioral and structural properties of the system. The first class concepts used in semantic modeling are Entity and Relationship. In our model both are present, but we have also user processes decomposition, like in DFD analysis, and more; the model includes conditional selection (like an if-then-else control structure) and functional application. Furthermore we promote the bottom up construction of the scheme opening the way to specification reuse.

Today OOA is the most promising industrial model to requirements analysis (see [dF92]). It moves up from the fast diffusion of the object oriented paradigm in the program languages and databases fields. But a proposal like Coad-Yourdon OOA[CY90], appears to be a remake of Yourdon's methodology, with an extension to include the new object concepts. No deductive capacity is possible from their model and the software engineer activity still remains pencil-paper work (with computer drawing facilities). Authors, like [HC91], propose a more formal model but intractable from the computational point of view. A complete formal model that has a reasoning technique and well founded semantics comes from A.Sernadas([SR91], [SF91]).

In the third area, KM, we find a lot of proposals extending a semantic model, usually E-R, with the temporal dimension and a knowledge representation method. Knowledge representation languages used in AI are able to express the same concepts of an E-R scheme, but they give better semantics to the concepts and they define other things that we don't have in an E-R scheme. The E-R extension can be found in the ESPRIT project ERT[LDa91] and ERAE[Dub88]. One of the most interesting projects in this field is the DAIDA environment (see [JMSV92]), that is built over the TELOS[MBJK90] language. The great expressive power of the knowledge representation languages, together with the conviction that requirements analysis is a first class expert activity, causes some authors to build expert systems as intelligent assistant for software engineers (see [BSo91]). Others show how to link databases conceptual scheme with concept languages (see [BS92], [dL89]). Few authors are dealing with the problem of express abstract or non monotonic reasoning in the data design (see [Bor88]).

DCM was proposed by A.Olivé in 1988[Oli89] and represents a pure deductive approach to the requirements analysis. In the model most of the effort is made to find the correspondence between the external and the internal model. The external model represents the declarative description of the user domain. The internal one is close to the deductive database behavior and defines which events insert or delete information. A drawback of this model is that the correspondence between the external and internal model is an expert task (see [San90]).

In LM area we find some attempt to export the semantic model developed for natural language understanding to software engineering. Sowa's pioneering work, Conceptual Graphs([Sow84]), has opened the way to other works in this direction, like [Dv91]. In this area most of the effort is made to have a model able to describe the meaning of the sentences used in the functional specification of the information system. A formal language that has an expressive power close to the natural language, mainly, gives us a computational intractable model. If we want an effective analysis tool we must reduce the language expressiveness to a subset of the full model.

3 Using the ERP Language.

The model presented in this paper starts from the assumption that the extension of the dynamic properties, pointed out in the analysis of the system functions, is completely determined by the knowledge of the structural system properties. The activities present in the user domain are viewed as processes that map system states into system states. Each state is completely described by the structural properties. The temporal dimension of the process is captured in a sense that is good enough for sequential and concurrent systems, but is not enough for real time problems. The UoD (Universe of Discourse), is divided in two parts. The extensional level refers to instances of individuals that populate the problem domain. The intensional level, the main subject of our investigation, is composed of a descriptive structure and a set of assertions. The descriptive structure, the ***skeleton***, defines the objects of interest and represent the alphabet of a logic language. The assertions define the properties of the objects and are terms of the logic language. The concepts used to describe the structural elements of the problem are: entities, relationships and roles. Each named entity is related to a class of domain objects. The relationships between entities are also named classes of objects. A relationship element is linked with a unique tuple of fixed arity. Each tuple component, one for each entity in the relationship, is an instances of the class. The roles define the link between an entity and a relationship. We denote with role(R)=$\langle U_1, \ldots, U_j \rangle$ the set of roles U for the relationship R. U_i means that the role U is present in the i-th place in R. We denote it also as $R_{U_1,\ldots,U_j}$. If E is the entity linked by U_i in R, then every instance r $\in$ R has as i-th component an element e of E, and e is identified by U_i. The assertions about a static schema are able to define the implicit properties of the class, ISA relationships between entities and entities and between relationships and relationships, static constraints and derived class. In the scheme it is necessary to represent explicitly the properties that are embedded in the domain representation. For each relationship R and for each role U of R, we must clarify the entity E that characterizes the role by the assertion:

$$(\exists R[U].U : \top_e) \sqsubseteq E$$

For example, referring to a lawyer office automation problem, the three assertions::

$$(\exists LegalDossier[Defendant].Defendant : \top_e) \sqsubseteq JuridicalPerson$$
$$(\exists LegalDossier[Plaintiff].Plaintiff : \top_e) \sqsubseteq Client$$
$$(\exists LegalDossier[Document].Document : \top_e) \sqsubseteq OfficeDocument$$

can be used to specify that the relationship $LegalDossier$ is defined, (in the role $Defendant$) on the entity $JuridicalPerson$, (in the role $Plaintiff$) on the entity $Client$ and (in the role $Document$) on the entity $OfficeDocument$. The ISA relationship can be directly represented as in the following assertions:

$$\begin{array}{r|l} Client \sqsubseteq JuridicalPerson & LegalAct \sqsubseteq OfficeDocument \\ LawyerDocument \sqsubseteq OfficeDocument & Notice \sqsubseteq OfficeDocument \end{array}$$
$$OfficeDocument \sqsubseteq (LawyerDocument \sqcup LegalAct \sqcup Notice).$$

The first assertion expresses an ISA relationship between two entities, as is usual in conceptual modeling. The other four assertions explain a richer potentiality of our language. We can express the closure of a generalized concept ($OfficeDocument$) over a finite set of classes. The ISA relationship can be established between relationships, also. We can have concepts represented by negative information, as:

$$\forall LegalDossier[Defendant].Defendant : \top_e) \sqsubseteq \neg Client$$

that express a professional principle: "a lawyer should refuse legal acts against his clients". Another possibility is to define derived classes through the declaration of necessary and sufficient conditions, as:

$$\begin{array}{lll} RelevantDocument & \equiv & (LawyerDocument \sqcap \\ & & \sqcap \forall LegalDossier[Document].Client : ImportantClient \end{array}$$

In general the possibility to express, in both parts of a scheme assertion, classes of individuals by complex expressions, give us a great expressive power.

3.1 Behavioral properties.

The dynamic aspect of the system is described by the behavior of the processes. We have two kinds of process: basic process and derived process. The definition of the former matches with its mapping declaration. The derived process is built using basic ones and the following constructors: conditional selection, functional composition, functional application. Every process must map only a relationship into a relationship or an entity into an entity. Is possible to overload a process, that is to use its name for different maps, but if a map is from an entity to an entity all the others have to be maps between entities (different entities, of course!). There is a particular class of basic processes predefined in the model: the projectors. The projector π_i is the map from a generic relationship of arity greater or equal to i to the entity that has the role U_i in the relationship. The activities present in the problem domain can be viewed as processes, that is a map between classes defined by the static scheme. We can follow two different ways to derive the process behavior. In a top-down style (decompositional), we refine a process describing its components, as:

$$
\begin{aligned}
&UpDateDossier : LegalDossier \rightarrow LegalDossier\\
&UpDateDossier.LegalDossier = UpDate_{DB}(UpDate \cdot \pi_1, UpDate \cdot \pi_2, UpDate \cdot \pi_3)\\
&UpDate : JuridicalPerson \rightarrow JuridicalPerson\\
&UpDate : Client \rightarrow Client\\
&UpDate : OfficeDocument \rightarrow OfficeDocument\\
&UpDate_{DB} : \top_R \rightarrow \top_R
\end{aligned}
$$

The process *UpDateDossier* is defined by the application of a basic process $UpDate_{DB}$ to several update processes, one for each relationship component. The defined process behavior expresses that we want the activation of a cascade of updating processes. How we do it is not relevant now. The updating will be obtained by an interactive or batch procedure, this will be fixed by the following steps. By example, if we want express a system requirement to fix the behavior, we can introduce an other basic process (eg.: *Interactive* or *Batch*) and declare the process type. The specific behavior will be defined in the design phase. The behavior of a process can be defined by the conditional selection, as in the following example:

$$
\begin{aligned}
&AddNotice : LegalDossier &&\rightarrow &&LegalDossier\\
&AddNotice.LegalDossier &&= &&ife(\forall LegalDossier[Defendant].Document : Notice,\\
& && && CallClient, SendNotice)\\
&CallClient : LegalDossier &&\rightarrow &&LegalDossier\\
&SendNotice : LegalDossier &&\rightarrow &&LegalDossier
\end{aligned}
$$

The process behavior description expresses the links between two different activities: *CallClient* and *SendNotice*. The office must send a notice to every defendant and must call the client after the notice dispatch. The condition characterize the meaning of the activity, without defining the specific behavior (telephone call, mail notice, etc.) or how the condition could be verified. Often in the specification analysis we find a situation similar to other problem already analyzed. In this case we would reuse the old specification. With our language the reuse is totally transparent, we simply include the old definitions and check the scheme consistency. It's important to remark that with our language operators we are able to express the time dimension of sequential and concurrent systems (see [CL91]). The functional application (eg.: $f(x,y,z)$) can be used to express the concurrency activation of several processes. The functional composition (eg.: $f \cdot g \cdot h$) can be used to express the concatenation of several processes. The conditional selection (eg.: $ife(c,y,z)$) can be used to express the activation, on an event C, of distinguished processes.

4 The ERP language: syntax.

A schema, subdivided in static and dynamic part, is a theory of a specific logic language. The language alphabet is named *skeleton* and is composed by the object name. The logic formulas are axioms of the theory.

4.1 Static Skeleton Σ_S

$\mathcal{R}$ is the set of relationship symbols, include $\top_R$ to denote the universal relationship. $\mathcal{E}$ is the set of entity symbols, include $\top_E$ to denote the universal entity. $\mathcal{U}$ is the set of role symbols. For each relationship $R \in \mathcal{R}$, $role(R)$ denotes the ordered tuple of the roles in R. For two entities A, B it may be asserted that: $A \sqsubseteq B, (A, B) \in exprR$ or $(A, B) \in exprE$. $exprR$, $exprE$ Set of *wwf* formed with the following syntax rules:

$$\begin{array}{lcl} P, Q & \rightarrow & R \mid P \sqcap Q \mid \neg P \mid \top_R \\ C, D & \rightarrow & E \mid C \sqcap D \mid \neg C \mid \top_E \mid \forall R[U].T_1 : C_1 \ldots T_n : C_n \mid \exists R[U].T_1 : C_1 \ldots T_n : C_n \end{array}$$

The idea is that all the knowledge about the basic elements in the *skeleton* can be specified in terms of a set of assertions, that forms the specification part of the scheme. **A**, **B** are expressions each one denoting a class (an entity or a relationship). Informally, an assertion of the form $A \sqsubseteq B$, states that every instance of the class denoted by the expression A is also an instance of the class B.

4.2 Dynamic Skeleton Σ_D

Π is the set of projector symbols. $\Pi = \{\pi_j\}$ each projector select a component of a relationship. $\mathcal{B}$ is the set of basic process symbols. $\mathcal{M}$ is the set of derived process symbols. For every process m it may be asserted that: $m : I_1 \rightarrow O_1, m : I_2 \rightarrow O_2, \ldots m : I_k \rightarrow O_k$, $m \in (\mathcal{M} \cup \mathcal{B}), (k > 0), (I_j, O_j) \in (exprE \cup exprR)$. The multiple definition of the same process grant the overloading, the type of the overloaded process is the union of the single types. For every derived process m it may be asserted that: $m : I_1 = def_1, m : I_2 = def_2, \ldots m : I_k = def_k$, $m \in \mathcal{M}, (k > 0), I_j \in (exprE \cup exprR)$. For each domain I_j in the type declaration we have a different behavior declaration: def_j. The whole process behavior is the union of the single definitions obtained by the following syntax rule:

$$\begin{array}{lcl} def & \rightarrow & defA \mid def \cdot defA \\ defA & \rightarrow & m' \mid \pi_j \mid if(C, def) \mid ife(C, def_t, def_e) \mid def'(def_1, \ldots, def_l) \end{array}$$

$m' \in (\mathcal{M} \cup \mathcal{B}), (C \in (exprE \cup exprR), \pi_j \in \Pi$, $\cdot$ denote the functions composition.

The idea for the dynamic scheme is that the behavior of the basic processes is given by the type declaration. The behavioral knowledge about derived processes can be specified by a set of assertions that explains the specific behavior. The $\cdot$ operator expresses the processes composition, this operator is useful to represent the sequential concatenation of activities. The functional application, $def'(def_1, \ldots, def_l)$, is able to describe concurrent activation of processes and ordinary composition as well. The conditional selection, if and ife, expresses the behavior of processes that realize different activities for different class of information.

5 The ERP language: semantics.

The language semantics is described in the model theoretic style and is reported in appendix. Now we want remark that an interpretation is a ***model*** for a scheme if the following conditions are verified. Condition 1: the static scheme assertions are valid to determine the classification hierarchies and then we can apply a subsumption algorithm to verify taxonomic properties. Condition 2: when we assert, in a dynamic scheme, that a process is of type $\beta : I \rightarrow O$, we say that β ought to be a map such that: if we active the process on an I element, then it returns an O element, at least. In a modal temporal logic, where we have an operator M to express obligation and an operator L to express possibility, an operator $[\beta]$ to express "must be after β" and an operator $\langle\beta\rangle$ to express "is possible after β", we can formally write the previous sentence as: $\neg M(I \wedge [\beta]\neg O) \equiv L(I \rightarrow \langle\beta\rangle O)$[1]. Condition 3: the set of elements pairs representing the extension of a derived process, contains all the elements pairs in the definition of the process, in accord to its type declaration. A process definition is valid, for a scheme, if it is verified in every model. The notion of model includes every interpretation that links the basic process symbols to total functions. This notion is used to exclude the possibility of an incomplete process definition due to its multiple descriptions. The interpretation of a dynamic scheme includes the models that links some classes of objects to the empty set. In this case we have a user domain concept that cannot be expressed. In the same way, we could have an empty process that represent an impossible computation. In this case impossible computation means that a process cannot get the necessary information. A process is, in our vision, not realizable if it doesn't find the information necessary to its activity. The process feasibility doesn't depend on the existence of an algorithm or of an automated task, but on the possibility to find the useful information into the problem domain. We consider this expressive level in order to represent one of the basic analysis concepts: the analysis must express "what" we can do and not "how" we can do it.

6 Reasoning about the Scheme.

During the requirements analysis, the designer often needs to perform several checks on the scheme being developed. Traditional analysis models lack formal methods for such activity. One notable feature of our model is the formal deduction method. The deductive power allow the software engineer to reason about all the meaningful aspects of the intensional level of the problem representation, in order to verify the correctness of the scheme. The reasoning method is obtained translating a requirements analysis scheme into a conjunction of Propositional Dynamic Logic (PDL [Gol87]) formulas. An assertion about the scheme is valid if the correspondent formula is satisfiable, it is not valid in the other case. The scheme translation formula is the conjunction of three subformulas. The first one, namely α, represents static and dynamic skeleton and it expresses the valid terms. The second one, formula β, represents the action that ought to be true in every reachable state. The last one represent the assertion to check. It is presented in the form of denial. If the conjunction of the above three formulas is satisfiable then the consistency check fail, otherwise the scheme is valid.

---

[1] Our intention is to give to the process the meaning of an activity that, obligatory, if it is realized give, at least, the expected result. This is close to our vision of the requirements analysis but is far from design point of view, where we want specify a process that must be realized. The design specification must grant the correct result for every process activation. Future works must show how transform requirements specifications to design specifications.

6.1 Scheme Translation.

Given a scheme Σ, constrained by a set of assertions $\mathcal{A}$, and given a set of type declaration for the basic processes $\mathcal{P}_b$, and a set of projectors Π, we can translate each derived process declaration σ_p, to the following formula: $\Phi(\Sigma, \mathcal{A}, \mathcal{P}_b, \Pi, \sigma_p) = \alpha \wedge \beta \wedge \gamma$ [2].

The formula β expresses the condition that every scheme assertion is satisfied in every reachable state. Every system evolution is described by a collection of states and a collection of accessibility relationships between states. The kind of relationships and the states internal structure differ from system to system. The goal of the requirements analysis is to obtain a full description of both states structure and accessibility relationships. The formula β is able to define the set of all reachable states and the set of all accessibility relationships such that in every state is satisfied the dynamic and static scheme assertions. The formula β defines the desired trajectories for the system. The formula γ expresses the derived process in terms of basic processes. The formula is able to check if the derived process definition is consistent with the scheme axioms. This grants that the process behavior doesn't bring the system in inconsistent states. Because we haven't recursive process definition the logic formulas are not really complex. This is intentionally made to 'keep the things simple', avoiding the aspects that transform the problem into an intractable one. It's known that the type checking for a general function, including overloading and recursive definitions, is a not decidable problem (see [AKW90]).

7 Some remarks and conclusion.

The system requirements specification needs tools able to support the engineer qualitative reasoning. The Computer Aided Software Engineering is the market answer to this necessity, but the most commercial CASE tools are able to follow the engineer activities only in a syntactic way. Few CASE products are able to represent and to manipulate knowledge about the problem domain. Our vision of a new generation of CASE tools is founded on logic. The logic must be used as a formal framework to define and characterize the problem knowledge. We refer to a useful logic that is computationally decidable. We believe that it is necessary to specify the class of requirements analysis concept constructors together with their complexity. Too many CASE products are delivered without having defined the trade off between necessary expressive power and computational tractable problem, and this is one of the motivations for the user expectation delusion.

In this paper we have proposed a formal semantics for the concept of process as it is usually found in the requirements analysis models. We believe that our definition shows two fundamental facts: 1) the qualitative reasoning is possible. In the sense that we can find a complete and decidable logic framework where the qualitative reasoning is expressed, 2) the computational class for the consistency check problem is EXPTIME in the worse case.This is an advice to don't believe that an easy solution to the whole problem exist and an invite to search different solutions for different classes of requirement analysis concepts.

This means that every CASE tool that is sealed as "able to reasoning over the schemes", probably don't meet this requirement (if the meaning of word reasoning is not correctly defined). But this means, also, that we have found a way to make truly reasoning CASE products.

---

[2] In [Rum93] is given a complete formal description of the translation scheme, together with a check example

A Appendix A.

The language semantics is described in the model theoretic style. Given a set $\Delta^{\mathcal{I}} = \mathcal{R}^{\mathcal{I}} \cup \mathcal{E}^{\mathcal{I}}$ of objects for entities and relationships, denoted with $^{\mathcal{I}} : (exprE \rightarrow 2^{\mathcal{E}^{\mathcal{I}}}) \cup (exprR \rightarrow 2^{\mathcal{R}^{\mathcal{I}}}) \cup (\mathcal{U} \rightarrow (\mathcal{R}^{\mathcal{I}} \rightarrow \mathcal{E}^{\mathcal{I}}))$ a mapping that links every entity or relationship description to a homogeneous set of $\Delta^{\mathcal{I}}$ objects, and every roles to a map from $\mathcal{R}^{\mathcal{I}}$ to $\mathcal{E}^{\mathcal{I}}$, and given a mapping $||^{\mathcal{P}} : ((\mathcal{B} \cup \mathcal{M}) \rightarrow (2^{\Delta^{\mathcal{I}}} \rightarrow 2^{\Delta^{\mathcal{I}}})) \cup (\Pi \rightarrow (2^{\mathcal{R}^{\mathcal{I}}} \rightarrow 2^{\mathcal{E}^{\mathcal{I}}}))$, that links process symbols to relationships in $\Delta^{\mathcal{I}}$, the triple $\mathcal{I} = (\Delta^{\mathcal{I}}, ^{\mathcal{I}}, ||^{\mathcal{P}})$ is an interpretation for a schema Σ if

α_0) For each role symbol $U \in \mathcal{U}$, $U^{\mathcal{I}}$ denotes the map $\mathcal{R}^{\mathcal{I}} \rightarrow \mathcal{E}^{\mathcal{I}}$, such that if $U \in role(R)$ and E is the entity associated by U, then for each $r \in R^{\mathcal{I}}$ we have $U^{\mathcal{I}}(r) \in E^{\mathcal{I}}$. If $U^{\mathcal{I}}(r) = e$ then we say that e is the U-th component of r and we denoted this as $r[\mathrm{U}]$.

α_1) For each relationship symbol $R \in \mathcal{R}$, $R^{\mathcal{I}}$ denotes a set of elements in $\mathcal{R}^{\mathcal{I}}$, each of them identifies, by the meaning of the roles in role(R), a unique tuple of elements in $E^{\mathcal{I}}$. In particular, if $R_{U_1 \ldots U_n}$ is a relationship defined in a schema Σ, then $R^{\mathcal{I}}$ is a set of elements $R^{\mathcal{I}} = \{r_1, \ldots, r_n\}$, such that for each r_j and for each role $U_k \in role(R)$ we have: $r[U_k] \in \mathcal{E}^{\mathcal{I}}$, $r[U_k]$ is in the set of objects interpreted by $^{\mathcal{I}}$ as the class E, and the class E is used, by role U_k, as k-th component of the relationship R[3].

α_2) For each entity symbol $E \in \mathcal{E}$, $E^{\mathcal{I}}$ is a set of elements in $\mathcal{E}^{\mathcal{I}}$.

β) For each static assertion the following equations are satisfied:

$$
\begin{array}{lcl|lcl}
(P \sqcap Q)^{\mathcal{I}} & = & (P^{\mathcal{I}} \cap Q^{\mathcal{I}}) & (C \sqcap D)^{\mathcal{I}} & = & (C^{\mathcal{I}} \cap D^{\mathcal{I}}) \\
\neg P^{\mathcal{I}} & = & \{r \in \mathcal{R}^{\mathcal{I}} | r \notin P^{\mathcal{I}}\} & \neg C^{\mathcal{I}} & = & \{e \in \mathcal{E}^{\mathcal{I}} | e \notin C^{\mathcal{I}}\} \\
\top_R^{\mathcal{I}} & = & \mathcal{R}^{\mathcal{I}} & \top_E^{\mathcal{I}} & = & \mathcal{E}^{\mathcal{I}}
\end{array}
$$

$$
\begin{array}{lcl}
\forall R[U].T_1 : C_1 \ldots T_n : C_n^{\mathcal{I}} & = & \{e \in \mathcal{E}^{\mathcal{I}} | \forall r \in R^{\mathcal{I}} r[U] = e \rightarrow (r[T_1] \in C_1 \wedge \ldots \wedge r[T_n] \in C_n)\} \\
\exists R[U].T_1 : C_1 \ldots T_n : C_n^{\mathcal{I}} & = & \{e \in \mathcal{E}^{\mathcal{I}} | \exists r \in R^{\mathcal{I}} r[U] = e \wedge (r[T_1] \in C_1 \wedge \ldots \wedge r[T_n] \in C_n)\}
\end{array}
$$

δ_0) For each projector symbol $\pi_j \in \Pi$, $|\pi_j|^{\mathcal{P}}$ is a binary relationship between elements of $\Delta^{\mathcal{I}}$, more precisely $|\pi_j|^{\mathcal{P}} = \{(r, e) | r \in \mathcal{R}^{\mathcal{I}} \wedge e \in \mathcal{E}^{\mathcal{I}} \wedge r[U_j] = e\}$, for each relationship R in Σ that has at least j components.

δ_2) For each basic process symbol $\beta \in \mathcal{B}$ and for each derived process symbol $m \in \mathcal{M}$, $||^{\mathcal{P}}$ denotes a binary relationship between elements of $\Delta^{\mathcal{I}}$, and more exactly:

$$
\begin{array}{lcl}
|m|^{\mathcal{P}} & \subseteq & \{(u, v) | u \in \Delta^{\mathcal{I}} \wedge v \in \Delta^{\mathcal{I}}\} \\
|\beta|^{\mathcal{P}} & \subseteq & \{(u, v) | u \in \Delta^{\mathcal{I}} \wedge v \in \Delta^{\mathcal{I}}\}
\end{array}
$$

γ) For each process definition the following equations are satisfied:

$$
\begin{array}{lcl}
|if(C, def)|^{\mathcal{P}} & = & \{(u, v) | u \in C^{\mathcal{I}} \wedge (u, v) \in |def|^{\mathcal{P}}\} \\
|ife(C, def_t, def_e)|^{\mathcal{P}} & = & \{(u, v) | u \in C^{\mathcal{I}} \wedge (u, v) \in |def_t|^{\mathcal{P}}\} \cup \\
 & & \cup \{(u, v) | u \in \neg C^{\mathcal{I}} \wedge (u, v) \in |def_e|^{\mathcal{P}}\} \\
|def'(def_1, \ldots, def_l)|^{\mathcal{P}} & = & \{(u, v) | \exists x_1, \ldots, x_l, r (\wedge_{j=1}^{l} ((u, x_j) \in |def_j|^{\mathcal{P}} \wedge r[U_j] = x_j) \\
 & & \wedge (r, v) \in |def'|^{\mathcal{P}})\} \\
|def \cdot defA|^{\mathcal{P}} & = & \{(u, v) | \exists t ((u, t) \in |defA|^{\mathcal{P}} \wedge (t, v) \in |def|^{\mathcal{P}})\}
\end{array}
$$

An interpretation is a *model* for a scheme iff: 1) for each assertion $A \sqsubseteq B$, of a static scheme Σ_S, the mapping $^{\mathcal{I}}$ links $A^{\mathcal{I}}$ to a subset of $B^{\mathcal{I}}$; 2) for each basic process symbol $\beta \in \mathcal{B}$, of a dynamic scheme Σ_D, if $\beta : I_1 \rightarrow O_1, \beta : I_2 \rightarrow O_2, \ldots, \beta : I_k \rightarrow O_k$ are all the type declarations of β then $||^{\mathcal{P}}$ is such that: a) for each type declaration: $\beta : I_j \rightarrow O_j$, if $u \in I_j^{\mathcal{I}}$ then exists at least a pair $(u, v) \in |\beta|^{\mathcal{P}}$ with $v \in O_j^{\mathcal{I}}$, b) if $(u, v) \in |\beta|^{\mathcal{P}}$ then $u \in \bigcup_{j=1}^{k} I_j^{\mathcal{I}} \wedge v \in \bigcup_{j=1}^{k} O_j^{\mathcal{I}}$; 3) for each derived process symbol $m \in \mathcal{M}$, of a dynamic scheme Σ_D, if $m : I_1 \rightarrow O_1, m : I_2 \rightarrow O_2, \ldots, m : I_k \rightarrow O_k$ are all m type declarations, and $m : I_1 = def_1, m : I_2 = def_2, \ldots, m : I_k = def_k$ are all m definitions, then $||^{\mathcal{P}}$ is such that: a) for each definition: $m : I_j = def_j$, if $u \in I_j^{\mathcal{I}}$ and $(u, v) \in |def_j|^{\mathcal{P}}$ then $(u, v) \in |m|^{\mathcal{P}}$, b) if $(u, v) \in |m|^{\mathcal{P}}$ then $u \in \bigcup_{j=1}^{k} I_j^{\mathcal{I}} \wedge v \in \bigcup_{j=1}^{k} O_j^{\mathcal{I}}$

---

[3] This definition is equivalent to require that the category formed by relationships, as objects, and roles, as arrows, is cartesian[Pie]

References

[AH87] S. Abiteboul and R. Hull. IFO: A formal semantic database model. *ACM Trans. on Database Syst.*, 12(4):297–314, December 1987.

[AKW90] Serge Abiteboul, Paris C. Kanellakis, and Emmanuel Waller. Method schemas. In *Ninth ACM Symp. on Principles of Database Systems*, pages 16–27, 1990.

[Bor88] A. Borgida. Type systems for querying class hierarchies with non-strict inheritance. Submitted draft, October 1988.

[BS92] S. Bergamaschi and C. Sartori. On taxonomic reasoning in conceptual design. *ACM Trans. on Database Syst.*, 17(3):385–422, 1992.

[BSG91] R.J. Brachman, P.G. Selfridge, and [illegible]. [illegible]: a knowledge-based software information system. *ACM Trans. on Off. Inf. Syst.*, [illegible], 1991.

[Che76] P.P. Chen. The entity-relationship model: Toward a unified view of data. *ACM Trans. on Database Syst.*, 1(1):9–36, March 1976.

[TL91] C. Theodoulidis and P. Loucopoulos. The time dimension in conceptual modelling. *Information Systems*, 16(3):273–300, 1991.

[CY90] P. Coad and E. Yourdon. *Object Oriented Analysis*. YOURDON PRESS, 1990.

[dCF92] D. de Champeaux and P. Faure. A comparative study of oo analysis methods. *Journal of Object Oriented Programming*, 1992.

[dBS89] [illegible]. A deductive model for o-o modelling. In *Fifteenth International Conference on Very Large Data Bases*, Amsterdam, 1989.

[Dub88] E. Dubois. Logical support for reasoning about the specification and the elaboration of requirements. In *Artificial Intelligence in Databases and I.S.* North Holland 1988.

[DvdR91] F. Dignum and R.P. van de Riet. Knowledge base modelling based on linguistics and founded in logic. *Data and Knowledge Engineering*, 7:1–34, 1991.

[EGH+92] G. Engels, M. Gogolla, U. Hohenstein, K. Hülsmann, P. Löhr-Richter, G. Saake, and H.-D. Ehrich. Conceptual modelling of database applications using an extended ER model. *Data and Knowledge Engineering*, 9:157–204, 1992.

[Gol87] R. Goldblatt. *Logics of time and computation*. CSLI, 1987.

[HC91] F. Hayes and D. Coleman. Coherent models for oo analysis. In *ACM Symp. on Object Oriented Programming Systems, Languages and Applications*, 1991.

[HK87] R.H. Hull and R. King. Semantic database modelling: Survey, applications and research issues. *ACM Computing Surveys*, 19(3):201–260, September 1987.

[JMSV92] M. Jarke, J. Mylopoulos, J.W. Schmidt, and Y. Vassiliou. DAIDA: An environment for evolving information systems. *ACM Trans. on Off. Inf. Syst.*, 10(1), 1992.

[Kun89] C.H. Kung. Conceptual modeling in the context of software development. *IEEE Trans. on Software Eng.*, 15(10), 1989.

[LC91] P. Loucopoulos, [illegible], and al. Design and execution of event/action [illegible] specification. In *II Int. Work. on the Deductive Approach to I.S. and DB.*, 1991.

[MBJK90] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Trans. on Off. I.S.*, 8(4):325–362, 1990.

[Oli89] A. Olivé. On the design and implementation of information systems from deductive conceptual models. In *15th Int. Conf. on Very Large Data Bases*, Amsterdam, 1989.

[Pie] B.C. Pierce. A taste of category theory for computer scientists. Technical report, Computer Science Department Carnegie Mellon University.

[Rom93] G. [illegible]. Analisi dei sistemi informativi: formalizzazione di aspetti dinamici e statici delle classi. Thesis, November 1993.

[San91] M.R. Sancho. Deriving an internal events model from a deductive conceptual model. In *2 Int. Work. on the Deductive Approach to I.S. and DB.*, 1991.

[Sch91] Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. of the 12th Int. Joint Conf. on Artificial Intelligence (IJCAI-91)*, Sydney, 1991.

[illegible]

[illegible]

[illegible]

ON BEHAVIOUR MODELING USING A FUNCTIONAL APPROACH

Jaroslav Pokorny

Charles University, Department of Software Engineering, 118 00 Praha 1, Malostranske nam. 25, Czech Republic

Abstract

In this paper we present a version of the HIT database model (see e.g. [Zla85], [Po89]) considering some behavioural features of modeling based on a three level approach. The proposal given here expresses the features using typed regular programs. A logic supporting data modeling capabilities of the HIT is embedded into a typed lambda calculus. This allows us to understand all structures and operations as functions. The approach is general enough to express different kinds of logical constructions, not necessary based on the first order language.

1. INTRODUCTION

The complete conceptual specification includes both the static and dynamic aspects of database applications. So called integrity constraints (IC) are conventionally used to define static and dynamic application properties that are not expressible using the object and operational features of a database model.

Concepts for modeling dynamic properties include primitive operations, control structures and dynamic constraints (or transition constraints) that cover situations where restrictions on sequences of database states must be imposed. When a violation of the constraints is detected, an exception handling mechanism is required.

Primitive operations are not adequate to represent directly each application operation. The operations lead to sequencing and, usually, a deep hierarchy of procedures. Recently data models have provided structural conceptual means for dealing with groups of operations. The behavioural abstractions are actions and transactions which are similar to procedures. The design and specification of transactions was intensively studied in the 80ties ([Br81],[Br82],[Br83]), and we can observe a reappearance of this effort now in connection with object-oriented design methodologies [Ho93]. A modeling of an active database behaviour now includes possibilities to generate rule/trigger definitions on a database level [Ja92].

The specification of an information system at three levels is close to the approach taken by Brodie [Br82], [Br84]. He presents behaviour modeling concepts as an [illegible] and structure. The resulting product of the **transaction level** is a specification of end user requirements such as application transactions, queries and reports. The **database level** involves the implementation of the specifications at conceptual and transaction level. A modification of this methodology with only one procedural abstraction - transaction scheme, is used in [Ng89].

Another approach is presented by Schiel et al. in [Sch84]. There the entire application design involves the formal application description which is divided into two intermediate levels. The state/transactions layer specifies the predicate types, static and dynamic ICs, the valid states and the valid transactions between states. As a lower layer, the operation level appears with appropriate update operations and tools for not violating static or dynamic ICs. The application modeling, as a conceptual level, serves as a representation of a formal application description using the called temporal hierarchical model [HS84].

A transaction can be expressed in a special kind of programming language. Programs in the language should preserve the consistency of a database. Well-known means for ensuring this property are pre- and postconditions that represent dynamic constraints ensuring the appropriate execution of operations [Br84], [HS84]. (Another structural methodology is introduced in [CG81].)

In the event that a precondition fails, exception handling mechanism techniques are essential in a complex application, but it is often difficult to use them.

In principle, there are several alternate approaches that can be used in a formalization of dynamic ICs. First, pre- and postconditions are specified in a structured manner directly in application oriented operations. The second approach uses a first-order language with explicit "state" and "time" parameters in terms of the languages. Finally, an extended temporal logic that does not refer explicitly to states and time is followed in [CF81], [Se80].

A formal logic of events and transactions is also introduced in [FS86]; variants of dynamic logic applied to programs (transactions) is given in [Ca81].

In this paper we present a version of the HIT database model (see e.g. [Po85], [Po90]) considering some behavioural features of modeling based on a three level approach. The proposal given here expresses the features using typed regular programs.

As an introduction to the problem, in Section 2 we review in intuitive terms HIT database model concepts. Section 3 rigorously defines the language of a typed lambda calculus that plays the role of a logical language. A formal framework related to the behaviour modeling concepts is presented in Section 4 and 5. Finally, Section 6 contains a language for specifying of a conceptual description of the system behaviour whose semantics can be expressed via typed regular programs. The final Section contains concluding remarks.

2. THE HIT DATABASE MODEL: A SHORT SURVEY

The HIT [illegible]

[illegible]

atomic objects, and functions called attributes. A database schema consists of a definition of all object types related to an application, including attributes and ICs.

The HIT DM supports usual kinds of atomic objects, i.e. abstract objects (for example employees, departments etc.), and descriptor objects.

An hierarchy of types is constructed as follows. The existence of some (elementary) types $S_1, \ldots, S_k$ ($k \geq 1$) is assumed. They constitute a **base B**. More complex types are obtained in the following way:

If $S, R_1, \ldots, R_n$ ($n \geq 1$) are types, then

(i) $(S{:}R_1, \ldots, R_n)$ is a **(functional) type**,

(ii) $(R_1, \ldots, R_n)$ is a **(tuple) type**.

The set of **types T over B** is the least set containing types from B and those given by (i)-(ii).

When S_i in **B** are interpreted as non-empty sets, then $(S{:}R_1, \ldots, R_n)$ denotes the set of all (total or partial) functions from $R_1 \times \ldots \times R_n$ into S, $(R_1, \ldots, R_n)$ denotes the Cartesian product $R_1 \times \ldots \times R_n$.

An important elementary type is Bool defined as the set {TRUE,FALSE}. The type Bool allows us to type such objects as sets and relations. Sets and relations are modelled as unary and n-ary characteristic functions, respectively.

The fact that X is an object of type $R \in T$ will be written alternatively X/R, X^R or "X is the R-object".

We don't include ISA-hierarchies defined on elementary types in the approach presented here. For later purposes, an existence of a universal elementary type U will be assumed. So, for each type $S \in B$ the statement "S is subtype of U" holds.

For example, mathematical functions may be easily typed. Arithmetic operations +, -, *, / are examples of (Number:Number,Number)-objects. Logical connectives, quantifiers and predicates are also typed functions, e.g., and/(Bool:Bool,Bool), R-identity $=_R$ is (Bool:R,R)-object, universal R-quantifier Π_R and existential R-quantifiers Σ_R are (Bool:(Bool:R))-objects. R-singularizer I_R/(R:(Bool:R)) denotes the function whose value is the only member of an R-singleton and in all other cases the application of I_R is undefined.

We introduce also the R(U)-identities [illegible] of type (Bool:U,R) for each elementary type, that allows us to compare objects of not equal types. For example, let 'three' be contained in [illegible]. The [illegible] is evaluated as FALSE and no type failure is recognized.

In the HIT DM so called **simple types** play a significant role:

(i) $(S{:}R_1, \ldots, R_n)$

(ii) $(\mathrm{Bool}{:}(S_1, \ldots, S_m), (R_1, \ldots, R_n))$

where $m, n \geq 1$, S, S_i and R_j are elementary or tuple types (except U).

We can observe that functional and relational objects can both be homogeneously viewed in a functional manner.

Associations among objects are specified by some functions called **attributes**. [illegible] approach, they [illegible]

[illegible] may be used to [illegible]

[illegible] Number, String, [illegible] reservations, [illegible] the base **B** contains besides the above types the abstract objects types Reserv (reservations), Crew (crews), Flight (flights), Employee (employees). The elementary type String will serve as a generic sort for names of employees, employees numbers and addresses of employees. As other description object types we choose Stime (start time), Ltime (landing times), Flnum (flight numbers), Date (dates), Prsid (person identifications), Des (destinations), Resnum (reservation numbers). Some subsets of Numbers correspond to salaries of employees, crew numbers and flight numbers.

Now we define the following attributes:

NE /String(Employee) /* name of employee */
AE /String(Employee) /* address of employee */
EMPNO/String(Employee) /* employee number*/
CST /Bool(Employee)(Crew) /* crew structure*/
CNO /Number(Crew) /* crew number*/
DASS /Des(Crew) /* destination assigned to crew */
FLRV /(Resnum,Flnum,Des,Date,Prsid)(Reserv) /* flight reservation */
OCCST/Number(Flnum,Des,Date) /* occupied seats */

Under certain assumptions, the candidate keys regarding the type object Employees are (NE, AE) or EMPNO. We specify the only one IO as:

Every employee is at most in one crew. /IO1/ □

3. THE TYPED LAMBDA CALCULUS

A version of the typed lambda calculus with tuple types directly supports manipulating objects typed by **T**. Starting with a collection Func of constants, each having a fixed type, and denumerable many variables of each type, the language of **(lambda) terms LT** consists of objects such as application and lambda abstraction. Moreover, if M is a term of type $(R_1,\ldots,R_n)$, then M[1],...,M[n] are terms of respective types $R_1,\ldots,R_n$. These term are called components.

Terms can be interpreted in a standard manner by means of an interpretation assigning to each function symbol from **Func** an object of the same type, and by a **semantic mapping [ ]** from **LT** into all functions given by the type system **T**.

Briefly said, an application is evaluated as the application of an associated function to given arguments, a lambda abstraction "constructs" a new function. In the conventional approach a valuation δ is used. This function allows us to assign objects to every variable occurring in a term. Practically, we can consider e.g. [+(3,x)] as a parametrized function which gives $3 + x$ for given x (more precisely, for every valuation). All valuations create a space VAL.

According to the semantics of the quantifiers and the singularizer, we can write an application of universal and existential quantifier simply as **foreach** x(M) and **forsome** x(M), respectively. Finally, we write I(lambda x(M)) shortly as **onlyone** x(M) and read "the only x such as M". (Naturally, M/Bool.)

Similarly, some aggregation operators such as COUNT_S/Number(Bool(S)), SUM/Number(Bool(Number)) etc. can be defined with usual meaning.

Occurrences of variables may be classified as **bound** or **free** in the usual way. We do not distinguish between terms differing only in the names of bound variables. By M[N/x] we will denote the substitution for every free occurrence of x in M by N, for M, N $\in$ **LT**.

For the purposes of a formal definition of basic database notions, we won't require an arbitrary language **LT**. We call a language **LT database-applicable** iff it has the following properties:
1/ Bool, Number and U are among the types of **B**.
2/ There is a distinguished binary predicate $=_R$ for any type
R among constants of **Func**.
3/ There is a distinguished binary predicate $\approx_R$, for any elementary type $R \in$ **B**, among constants of **Func**.
4/ There are logical connectives, R-quantifiers, and R-singularizer (for any R) among constants of **Func**.

All other LTs may be considered as an extension of a given database-applicable language **LT**. We will assume a fixed interpretation φ assigning to symbols of **Func** in **LT** their intended interpretation. A **functional model** $\mathbf{M}_{LT}$ of **LT** is a couple (φ, [])
In the remainder of the paper, we consider database-applicable languages **LT** only.

4. HIT DATABASE CONCEPTS

First, it is necessary to discusse some details of the LT language used in the HIT approach. Note that the connectives are given not by their usual truth table definition but rather as partial functions. This is equivalent to an extension of the conventional truth values of the type Bool with a third truth value expressing "undefined". Thus, contrasting with standard approaches (see, e.g., [Co79]), a different interpretation of e.g. disjunction is used; [TRUE v "undefined"] under each valuation δ gives the value "undefined" here. This fact affects the understanding of database queries.

It is possible to alternate these semantics in some cases. For example, so called **weak equality** $=_w$ can be considered, where $M =_w N$ is defined as follows:
TRUE if [M = N] gives TRUE, and
FALSE if [M = N] gives FALSE or "undefined
under a valuation δ.

Let **LT** be a database applicable language. Then a **(HIT-) conceptual schema** (with respect to **LT**) is a couple $\mathbf{S}_{LT}$ = **(A,I)**
where **A** is a finite set of simple typed variables of **LT** called attribute identifiers and **I** is a set of other terms of **LT** representing IC. The last terms are of type Bool.

Note that each conceptual schema $\mathbf{S}_{LT}$ is a term of **LT**. It is reasonable to restrict ICs to terms with attribute identifiers as the only free variables.

Let **A** = $(A_1, \ldots, A_K)$, $K \geq 1$, and δ be a valuation. A (**HIT-**) **information base** of $\mathbf{S}_{LT}$, is the tuple $(\delta(A_1), \ldots, \delta(A_K))$
We denote it as $\mathbf{S}^*$ whenever we assume a fixed language **LT**. A $\mathbf{S}^*$ is consistent iff [I] = TRUE under δ for all $I \in$ **I**. The valuation δ restricted to A_i, for $i \in <1,K>$, is often called an **information base state**. The valuation δ corresponding to a consistent information base is called a **consistent state**.

The definition of an information base can be naturally structured into four divisions. First, the definition of description object types in an appropriate string definition language (see e.g. Beta [Br82]) is necessary. Second, the definition of abstract object types is useful with a conceptual aggregation associating the given type for all attributes defined on it. For example, the abstract type *Employee* may be conceptually understood as an abstraction, components of which are attributes. Thus, the elementary type Employee represents domain objects for attributes NE, AE and EMPNO. Some subset of the attributes is specified as the total object key. Third, the attributes expressing functional associations among objects are defined, i.e. attributes identifiers and their respective types are given. Finally, a facility for expressing ICs based on an appropriate subset of **LT** completes a HIT-conceptual schema.

It is also possible to deal with "inversions" of attributes on the conceptual level. For example, given an employee object, we may apply a reversed attribute "Crew of an employee" to obtain a crew of the employee. It is easy to specify this reversed attribute by an appropriate term of lambda calculus **lambda** e **onlyone** c ((CST(c)(e)).

The result is of type Crew(Employee). We can conclude that a certain lambda term can always be used for a specification of a derived structure. We can assume for any object key $(K_1, \ldots, K_s)$, $s \geq 1$, their reversed derivates $(KI_1, \ldots, KI_s)$ accessible in each conceptual schema.

5. TYPED REGULAR PROGRAMS

In this section we introduce a language LTRP appropriate for accessing HIT-information bases described by conceptual schemes. The programs of LTRP are similar to regular programs considered in [CF83]. We call them **typed regular programs** (TRP). The language LTRP contains constructs which permit us to define semantics for user-oriented syntax of a data manipulation language (DML) over HIT-information base. The expressive power of TRPs is sufficient to express such control structures as e.g. in [Br84], [HS84].

Usually four primitive operations are considered in a DML: insert, deletion, and update for modifying objects in an information base, and retrieve for selecting objects. Instead of the operations we will define the typed assignment x := M, where x is a variable and M is a term of **LT**. Both x and M are of the same type.

A test M^{Bool}? indicates whether or not a computation should continue. A TRP is built, as usual, by three structuring principles - conjunction, disjunction, and iteration, i.e. either its statments are executed sequentially (conjunction), or only one of them is selected for execution (disjunction), or else its statements are invoked in a loop (iteration).

Let **LT** be a database-applicable language of terms. The **language of typed regular programs** (LTRP) (over **LT**) is defined as follows:

basic programs:

(bi) if x and M are a variable and a term of LT, respectively, of type S, then x :=M is in LTRP /**assigenment**/

(bii) if M is of type Bool, then M? is in LTRP /**test**/

composite programs:

if p and q are in LTRP, then

(ci) (p or q) /**disjunction**/

(cii) (p and q) /**conjunction**/

(ciii) q* /**iteration**/

are in LTRP. We say that s is a **component** of q iff s is q, or s is of the form (bi)-(ciii), or s is a component of a component of q.

The meaning of programs from LTRP with represent to the functional model is given by a function μ: LTRP $\rightarrow 2^{VAL \times VAL}$ in the following manner:

(bi) $\mu(x := M) = \{\alpha,\beta \mid \alpha,\beta \in VAL$ and $\beta = \alpha[M/x]\}$
(bii) $\mu(M?) = \{\alpha,\alpha \mid \alpha \in VAL$ and $[M] = TRUE$ under $\alpha\}$
(ci) $\mu(p$ or $q) = \mu(p) \cup \mu(q)$ /**union**/
(cii) $\mu(p$ and $q) = \mu(p) \circ \mu(q)$ /**composition**/
(ciii) $\mu(s^*) = (\mu(s))^*$ /**reflexive and transitive closure of** $\mu(s)$/
where p, q are from LTRP, and $\circ$ denotes a relational composition.

Note that besides the sequential executional semantics, LTRP allows us to use alternative semantics, either executional or declarative. Moreover, for example in (cii), a concurrent or nondeterministic sequential execution can be required.

If p is in LTRP, then the sets $\{\alpha \mid (\alpha,\beta) \in \mu(p)\}$ and $\{\beta \mid (\alpha,\beta) \in \mu(p)\}$ are called input and output states of p, respectively. Clearly, states (i.e. valuations) are not restricted to populations of attributes. A TRP contains possibly more variables.

Now, a program p over LT is said to **diverge** in the functional model $\mathbf{M}_{LT}$ iff $\mu(p)$ returns an empty set. Usually we are interested in such programs from LTRP that **converge** for any initial input state. Thus, given an initial input state α_0 a converging program q implies an existence of a sequence $(\alpha_0,\alpha_1,\ldots,\alpha_t)$, so called a **transition history**, such that a state α_i, $0 \leq i < t$ corresponds to each component p of q, and $(\alpha_i,\alpha_{i+1}) \in \mu(p)$. Conversely, when α_i, $i<t$, is in the transition history, then there are two components r and s of q such that α_i is an input state of r and such that α_i is an output state of s. If the transition history is given by a program q from LTRP uniquely for any initial state we say that q is deterministic. If the transition history contains only consistent states, then q **preserves consistency** of the information base.

In the case when the value of test M? is "undefined", it is useful to extend the semantics of TRP by explicitly stating $\mu(\text{"undefined"}?) = \{\phi,\phi\}$.

Now usual constructs known from high-level programming languages can be defined.

(i) Syntax: if M then p else q otherwise warning(<text>)
Semantics: (((M? and p) or ((not M)? and q)) or ("undefined"?))
(ii) Syntax: if M then p
Semantics: ((M? and p) or (not M)?)
(iii)Syntax: while M do p
Semantics: $(M?$ and $p)^*$ or (not M)?)

where warning(<text>) denotes a print operation call with a warning message returned not affecting states. Parentheses begin and end may be used in the usual way.

6. BEHAVIOUR MODELING IN HIT DM

According to the ACM/PCM methodology, we will distinguish three levels of behaviour modeling in HIT DM.

The purpose of the first level is to define a set of primitive **database operations** which support manipulation with functions of simple types occurring in the conceptual schema $\mathbf{S}_{LT}$. Semantics of these operations is given by decision of the designers in the implementation level. With respect to the $\mathbf{S}_{LT}$, we can formally assume corresponding function symbols for **Func** introducing these operations. Note that database operations may not preserve consistency of a given database. Database operations work on the internal level where additional implementation assumptions are necessary. However, the property of consistency is the conceptual one.

Behaviour modeling at the conceptual level involves the design and specification of actions for each abstract object and attribute. An action is a behavioural property of one object (in the Brodie sense - see [Br84]) and can be invoked to produce a state transition for the object (i.e. transition of the whole database).

A transaction is designed for a specific application over the conceptual level, i.e. it is composed of actions and/or transactions. Transactions produce transition histories. These sequences of states are often called events. Semantics of actions and transactions will be expressed by corresponding typed regular programs. Both the type manipulations can be modelled as specification schemes or as action and transaction "programs", respectively. The former case includes stating of input/output parameters and formulating of pre (post)-conditions. In the latter case, a "procedure body" is defined that plays an active role. We will follow this way of the solution without explicitly defining postconditions in the first step of the design.

All three types of manipulation (database operations, actions, and transactions) can be started explicitly by users or implicitly as inner calling of other manipulations. Also, it is necessary to build a parameter mechanism in a hierarchy of invocations.

The database operations allow the user to **create** abstract objects, i.e., a corresponding system procedure evokes the existence of the object of the given object type. The existence of description objects is assumed. The new object can only be accessed via the reference returned.

The **drop** operation ends the existence of an object given by its input parameter. An object cannot be deleted unless all function links have been previously deleted. Finally, a type checking is recommended. We consider the - **isobject-oftypeS** operations returning the value TRUE if the input object reference denotes an exiting object of the given type S. Parameters of such operations will be variables typed by the type U. It implies a possibility for accepting some badly typed objects. The recognition problem for these failures is propagated into the **isobject-oftypeS** operation.

Note that because our language **LT** has no parametrized types we introduce database operations for each elementary type. In the following the operation definitions are given by their syntax.

{**create** | **drop** | **isobject** } (<par>) **oftype**<btype>

The parameter is a variable of a given type from **B**. It acts as the output and input parameter in create and delete operations, respectively.

We will develop no exception handling mechanism in this paper. In the case that a precondition fails the user can call the database operation warning(<text>) and only a warning message is returned. Then the whole manipulation involving this operation is cancelled. The <text> is an arbitrary 'text' assigned automatically to a system variable.

The predefined operations over attributes ensure to define (undefine and redefine) a value of an attribute for a given object (or tuples of objects). The domain objects of corresponding types must exist. In the case of an incorrect typing the operation is equivalent to the warning operation.

The syntax of the **establish** operation is of the form:

establish forob <in-dparlist> **valueof** <pa>**with** <pa> (<in-dparlist>) = <in-rpar>;
<in-dparlist> ::= <in-dpar>[,<in-dpar>]* <in-dpar> ::= <var> | <term>
<in-rparlist> ::= <in-par>[,<in-par>]* <in-rpar1> ::= <var>
<in-par> ::= <var> | <term>
<in-rpar> ::= <in-rpar1> | (<in-rparlist>) | **set** [(<in-rparlist>)]

where <var> and <term> are a variable and a term of LT, respectively. The only <pa>

is both an input and output parameter of the operation. The remaining parameters are the input parameters.

Example 2: The definition of the establish operation for the attribute FLRV might be of the form: **establish forob** r **valueof** FLRV **with** FLRV(r) = (rn,fn,de,da,pr);

The value returned by the establish operation called on the actual arguments results in the changed value of the FLRV attribute. □

Similarly, the operation **detach** deletes a function link, and the **update** operation is used to modify the value associated with objects in the <in-dparlist>. Note that the semantics of the three kinds of operations is expressible by trivial TRPs.

An action specification at the conceptual level must ensure the consistency of the database. This is achieved by preconditions. The fundamental kind of action is to **originate** an abstract object. To do this, we must create it and define values of its object key, i.e. to establish values of certain attributes.

Further, the conceptual equivalents to the **establish**, **detach**, and **update** operations are defined. They are called **insert**, **delete**, and **modify**, respectively. An action is defined by its name, parameters, preconditions, and specified sequences of operations (actions) calls implied by test results. The parameters in the definition are of the local, input and output mode, respectively. A parameter value can be generally given by a term of **LT**.

Example 3: Let the object key of Employee object type be the attribute EMPNO. As an example, consider the action of inserting a new crew (parameter c) consisting of a given group of employees (parameter x). Its specification is given by the **where** clause in the following definition

```
action insert forob c value CST as x where
if not (isobject c oftypecrew) then warning('type failure in c')
else   if forsome y/U (x(y) and  not isobject y oftypeemployee )
       then warning('type failure in x - a set of  nonemployees')
       else if CST(c)=x  then  warning('the crew exists')
                         else   if not(foreach c1/Crew(foreach e/Employee(
                                  (CST(c1)(e) and not(c1=c)) implies  not CST(c)(e) )))
                                then warning('crews overlap')
                                else establish forob c valueof CST with CST(c) = x;
```

Note, that x/Bool(U), i.e. only the type failure in employees can be announced. A bigger "structural" failure in the value of x is displayed generally by a standard type control mechanism. The hierarchy of **if-then-else** constructs constitutes the preconditions ensuring a successful execution of the establish operation. □

Generally, preconditions determine whether some other actions must be invoked. For example, in the inserting FLRV attribute it is necessary to call the **update** operation for OCCST ensuring that the assignment n:=n+1, where n contains the number of occupied seats, is performed.

The fourth **if** in the Example 3 activates the IO1 (see Example 1). All parameters are typed and play a role of variables in associated **LT**. Input and output parameters are understood for a context. The local parameter is, e.g., the variable c1. It is straightforward to show that actions are easily transformable into TRP programs.

Finally, transactions have a structure similar to the action definition. They access objects by means of their actions only with the help of some preconditions test.

Example 4: Let CNOI and EMPNOI be the reversed derivates to CNO and EMPNO, respectively. The following transaction establishes for a given crew number the group of corresponding persons represented by their employee numbers.

```
transaction crew-establishment for
  domain x oftype Number
  range y oftype Bool(String)
  where insert forob CNOI(x) value y as
        lambda z/Employee(forsome g/Number(EMPNOI(g) = z and y(g) ))
```

where x and y are input and output parameters, respectively.

7. CONCLUSION

The designed methodology and formal apparatus are appropriate tools for a formal behavioral specifications in the HIT DM. A future research work might focus on extending the corresponding languages to temporal aspects needed in conceptual modeling, such as time-intervals, "before" relations, etc. The language manipulating conceptual objects can be used as the basis of an object oriented environment.

REFERENCES

[St93] Stein,W.,(1993), Objectoriente Anylysemethoden - ein Vergleich, *Informatik Spektrum*, Band 16, Heft 6, pp. 317-332.

[Br81] Brodie,M.L.,(1981), Data abstraction for designing database-intensive application, Proc. Workshop on Data Abstraction, Databases, and Conceptual Modeling (Eds. M.Brodie and S.N.Zilles). SIGPLAN Notices 16,1.

[Br82] Brodie,M.L.,(1982), Axiomatic definitions for data model semantics, *Inf. Syst.*,7,2, pp.183-197.

[Br84] Brodie,M.L., and Ridjanovic,D.,(1984), On the Design and Specifications of Database Transactions, *in*: "On Conceptual Modeling: Perspectives for AI, Databases, and Programming Languages", M.Brodie et al, eds., Springer - Verlag.

[Ca83] Casanova,M.A.,(1980), A Formal System for Reasoning about Programs Accessing a Relational Data, ACM *TODS*, 2,3, pp.386-414.

[Ce81] Ceri,S. et al,(1981), Structured methodology for designing static and dynamic aspects of database applications, *Inf. Systems*, 6,1,pp. 31-45.

[CF83] Casanova,M.A., and Furtado, A.L., (1983), On the description of database transaction constraints using temporal language, IBM Report RT 008.

[Co79] Codd,E.F.,(1979), Extending the Database Relational Model to Capture More Meaning, *ACM TODS*, 4,4, pp. 397-435.

[DE92] Bulletin of the Technical Committee on Data Engineering, Dec., 1992, IEEE Comp. Soc.

[DKMS86]Duží,M., Krejčí,F., Materna,P., and Staníček,Z.,(1986), HIT Method of Data Base Design: a Functional Approach to Information Representation, RR., Techn. Univ. Brno.

[DM90] Duží, M., and Materna,P.,(1990), Attributes: Distinguishing Capability versus Information Capability, *Computers and AI*, Vol. 9., No. 2, pp. 169-185.

[FS86] Fiadeiro,J., and Sernadas,A.,(1986), The INFOLOG linear tense propositional logic of events and transactions, *Inf. Systems*, 11, 1, pp. 61-85.

[HS84] Hordasch,A., and Schiel, U.,(1984), THM/CSL: A Language for conceptual schema design, MS, Institut fuer Informatik, INI Stuttgart.

[Ng89] Ngu, A.H.H.,(1989), Conceptual Transaction Modeling, *IEEE Trans. on Knowledge and Data Engineering*, Vol. 1, No. 4.

[Po89] Pokorný, J.,(1989), A function: unifying mechanism for entity-oriented database models, ER Approach, C.Batini (Ed.), Elsevier Science Publishers B.V. (North-Holland), pp. 165-181.

[Po93] Pokorný J.,(1993), Semantic Relativism in Conceptual Modeling, DEXA'93, Proc. of 4th Int. Conf. on Database and Expert Systems Applications, Prague 1993, Springer-Verlag.

[Sa80] Sardenas,A.,(1980), Temporal aspects of logical procedure definition,*Inf.Syst.*,5,3, pp.167-188.

[Sch84] Schiel,U. et al,(1984), Towards multi-level and modular conceptual schema specifications, *Inf. Systems*, 9,1, pp. 43-57.

[Zl85] Zlatuška,J.,(1985), HIT Data Model. Database from the Functional Point of View, VLDB'85, Stockholm.

A FRAMEWORK FOR CONCEPTUAL EXCHANGE BETWEEN OBJECT-ROLE MODELLING AND EXTENDED ENTITY-RELATIONSHIP MODELLING

Phillip M. Steele[1] and Arkady B. Zaslavsky[1]

[1] School of Computing & Information Technology, Peninsula Campus, Frankston, Monash University
McMahons Road, Frankston, Victoria 3199, Australia; Email: {PSteele, AZaslavs}@Monash.edu.au

Abstract

Although a wide variety of data modelling techniques exist, relatively little attention has been paid to the problem of their integration and ways of supporting interoperability amongst new and existing approaches. Whilst the entity-relationship approach is comparably well established and developed, object-role modelling is rapidly growing and gaining wider popularity. This paper outlines an attempt to develop a framework to integrate semantically and syntactically different modelling techniques using object-role modelling and extended entity-relationship modelling as a testbed. It is suggested that the approach used could be extended to provide a more general basis for the integration of many existing modelling techniques covering both static and dynamic aspects of applications. This research forms part of the DROTIS project, which aims to develop a generic meta model, and a set of interfaces and protocols to support the integration and interoperability of various system modelling techniques and CASE tools.

1. BACKGROUND

A wide variety of information system development methods exist utilising different system modelling techniques and life cycle models. These methods guide the system developer in constructing a set of models of the application domain, which are later transformed into models representing the design of the computerised information system. Each of these models is expressed using a particular modelling technique, such as functional decomposition diagrams, dependency diagrams, data flow diagrams, structure charts, flowcharts, action diagrams, state-transition diagrams, data structure diagrams, entity-relationship diagrams or object-role models (Martin & McClure, 1985; Nijssen & Halpin, 1989).

The latest paradigm shift to object orientation has introduced a whole new range of modelling techniques and diagram types, with few attempts to keep compatibility with previous approaches, or to provide guidelines on how to move from techniques such as entity-relationship modelling, object-role modelling and data flow diagraming to emerging ones suitable for object-oriented design (Beynon-Davies, 1992).

A number of researchers argue that this lack of interoperability serves to constrain the benefits derived from existing and newly emerging system modelling techniques and CASE tools (Atzeni & Torlone, 1991). A comparison of meta models for entity-relationship and object-role modelling techniques based on their expressive power, convenience, efficiency and learnability is discussed by Halpin & Oei (1992). An interesting framework for information systems concepts based on meta model hierarchy is suggested by Oei et al (1992). Several groups are working to define standards, which will allow CASE tools to exchange information between models expressed using variations of commonly used notations (EIA, 1991; ECMA, 1990; Imber, 1991). Researchers at the American National Standards Institute are also investigating how to integrate a wider variety of heterogeneous system modelling techniques (ANSI, 1992).

This paper outlines an approach to integrating object-role modelling (ORM) and extended entity-relationship (EER) modelling. These techniques were chosen because they are both data modelling techniques that appear to have significant conceptual and notational differences (Laender & Flynn, 1993), and because they are both supported by CASE tools, which simplifies the definition of their meta models.

A meta model of each modelling technique has been developed and expressed using the modelling technique that it is defining. The question of the ability of a CASE tool to express its own meta model is interesting *per se*. If it is possible to prove that an expressive power of a particular modelling technique is enough to represent a comprehensive meta model expressing all the static and dynamic aspects of another modelling technique, then it might be stated that the first modelling technique semantically supersedes the second one. While the search for one "universal" modelling technique is admirable, an alternative approach is to determine which existing system modelling techniques may be used in conjunction to represent all required static and dynamic aspects of applications. It is a separate research problem to develop a minimal set of such modelling techniques.

In an attempt to advance further towards the integration of modelling techniques, ORM and EER meta models have been expressed using the meta model formalism suggested by Atzeni & Torlone (1991), because it is generic in nature, and has been developed specifically for the purpose of integrating data modelling techniques. A number of extensions to their work have been proposed in the paper to support the integration of both approaches to data modelling.

It is recognised that meta models expressed using this approach alone, have insufficient power to represent all the semantics of any given modelling technique. However, representing only the static aspects of the selected modelling techniques still provides a useful, albeit incomplete, insight into interoperability issues. While the graphical aspects of modelling techniques are not trivial, they are not considered central to the semantics of modelling techniques themselves.

2. EXTENDED ENTITY-RELATIONSHIP MODELLING

The approach to extended entity-relationship modelling examined in this paper is the one used by the Texas Instruments' Information Engineering Facility (IEF™) CASE tool that supports a number of life cycle stages including information strategy planning, business area analysis, business system design, technical design and construction.

Entity-relationship modelling may be performed during information strategy planning and/or business area analysis. A simplified meta model for the EER approach used by the tool is included at Figure 1 (Steele & Zaslavsky, 1993b). The meta model has been represented using the IEF tool itself.

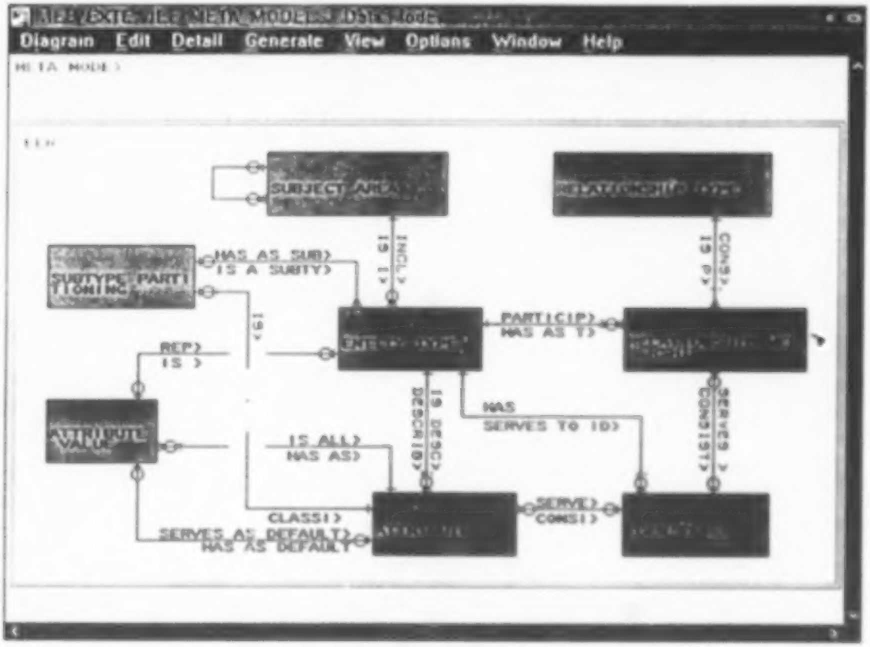


Diagram
Edit
Detail
Generate
View
Options
Window
Help
SUBJECT AREA
SUBTYPE PARTITIONING
ATTRIBUTE VALUE
HAS AS SUB>
IS A SUBTY>
IS I>
INCL>
IS P>
CONS>
IS>
REP>
IS >
PARTICIP>
HAS AS T>
IS DESC>
DESCRIB>
HAS
SERVES TO ID>
SERVES >
CONSIST>
IS ALL>
HAS AS>
CLASSI>
SERVES AS DEFAULT>
HAS AS DEFAULT
SERVE>
CONSI>

Figure 1. Meta model of the EER approach (Steele & Zaslavsky, 1993b)

An attribute of an entity type may be used to create a super-type partitioning into a number of other entity sub-types. Specific values of this partitioning attribute in the super-type, may be linked to each of the sub-types defined in the partitioning.

An entity type may participate in zero or more relationships and each relationship has two memberships with each membership linked to each entity type participating in the relationship. The limit of two is not specifically enforced with the EER model shown, which will in fact support n-ary relationships as well as the binary relationships supported by the IEF. Each entity type may have a number of identifiers with each identifier consisting of one or more attributes of that entity type and/or one or more relationship memberships associated with that entity type. One of the identifiers is flagged as primary. Each attribute describes only one entity type and may have one default value and number of allowable values.

This meta model, in conjunction with detailed definitions of its components defines the particular approach to EER modelling used by the IEF in considerable detail.

However, it is not generic in nature, and cannot be easily compared to meta models of other approaches, which are expressed using different modelling techniques. The meta model has therefore been expressed using the formalism suggested by Atzeni & Torlone (1991), which provides a more generic set of concepts to define a wide variety of data modelling approaches. This meta model is shown in Figure 2.

```
MODEL Texas Instruments, IEF, Extended-Entity Relationship

CONSTRUCTS

LEXICAL                  domain:
LEXICAL                  constraint-code:
ABSTRACT                 entity-type  (KEY identifier):
N-ARY AGGREGATION        relationship-type ON entity-type  (KEY key-entities):
UNARY MULTIVALUED FUNCTION
                         relationship-membership  FROM entity-type TO relationship-type
N-ARY MULTIVALUED FUNCTION
                         constraint FROM  relationship-membership TO constraint-code
UNARY MONOVALUED FUNCTION
                         attribute FROM entity-type TO domain:
N-ARY MONOVALUED FUNCTION
                         identifier FROM attribute, relationship-membership TO entity-type
UNARY MONOVALUED FUNCTION
                         super-type-partition FROM entity-type TO attribute
N-ARY MONOVALUED FUNCTION
                         super-type-partition-value FROM super-type-partition, domain TO entity-type
```

Figure 2. Generic meta model of the extended entity-relationship approach

An entity type is defined as an abstract concept, while a relationship type is defined as an aggregation of entity types, and a relationship-membership is defined as a function from one of the participating entity types in a relationship-type and the relationship-type itself. One or more constraints, such as cardinality, deletion semantics and transferability may be attached to each relationship-membership, through a multivalued function.

3. OBJECT-ROLE MODELLING

The approach to object-role modelling examined in this paper is that used by the Serverware InfoDesigner database design tool, and is based on the object-role approach as defined by Nijssen & Halpin (1989). The tool supports the definition of conceptual schema diagrams, and is able to generate relational database definitions with appropriate rules, procedures and triggers to ensure enforcement of the constraints defined in the conceptual schema. A simplified meta model for the object role approach used by the tool, represented using the InfoDesigner CASE tool itself is included at Figure 3. It should be noted that the version used for this comparison, is a superseded pre-release version that has not yet implemented all of the constraint types specified in the tool documentation.

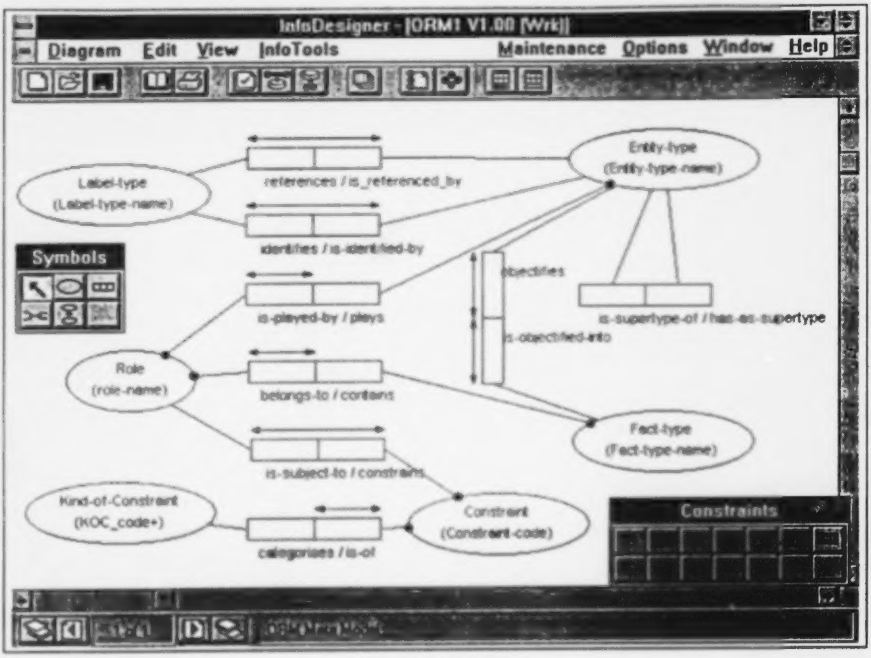

InfoDesigner - [ORM1 V1.00 [Wrk]]
Diagram
Edit
View
InfoTools
Maintenance
Options
Window
Help
Entity-type
(Entity-type-name)
Label-type
(Label-type-name)
references / is_referenced_by
identifies / is-identified-by
Symbols
objectifies
is-played-by / plays
is-supertype-of / has-as-supertype
is-objectified-into
Role
(role-name)
belongs-to / contains
Fact-type
(Fact-type-name)
is-subject-to / constrains
Kind-of-Constraint
(KOC_code+)
Constraint
(Constraint-code)
Constraints
categorises / is-of

Figure 3. InfoDesigner meta model of the object-role approach

Fact types consist of a collection of roles with each role participating in only one fact type. The position of the role in the fact type has not been not modelled in order to simplify the meta model and keep each fact type binary. Each role is always played by an entity-type and each entity-type must play at least one role, but may play many. In order to simplify the meta model, only a simple entity identification and referencing scheme has been allowed. A label type may serve to reference many entity types and an entity type may be referenced by many label types. One of these label types will serve to identify each entity type. Further research is required to support a more general entity identification scheme. Entity types may be super-types or sub-types of others establishing a super-type / sub-type hierarchy. Fact-types may be objectified to become composite entity types. Each constraint is of a certain type, and a constraint serves to limit the population of one or more roles.

The simplified meta model defined using the InfoDesigner CASE tool and shown in Figure 3, may also be expressed using the formalism suggested by Atzeni and Torlone (1991), in order to facilitate the matching of object-role concepts to extended entity-relationship concepts. This definition of the meta model is shown in Figure 4.

The entity-type is defined as an abstract concept, while the fact-type is defined as an aggregation of entity-types. A role is defined as a function on a fact-type and one of the entity-types participating in that fact-type. A type hierarchy is defined as a function from one or more entity-types (super-types) to one or more other entity types (subtypes).

EST COPY AVAILABLE

A constraint is defined as a function from a role to the lexical construct, kind-of-constraint.

MODEL Serverware, InfoDesigner, Object Role Approach

CONSTRUCTS

LEXICAL	label-type:
LEXICAL	Kind-of-Constraint-code
ABSTRACT	entity-type (KEY identifier)
N-ARY AGGREGATION	fact-type ON entity-type (KEY entity-type-name)
UNARY MONOVALUED FUNCTION	role FROM entity-type TO fact-type
N-ARY MULTIVALUED FUNCTION	constraint FROM role TO Kind-of-Constraint-code
UNARY MONOVALUED FUNCTION	entity-identifier FROM entity-type TO label-type
UNARY MONOVALUED FUNCTION	reference-mode FROM entity-type TO label-type
N-ARY MULTIVALUED FUNCTION	type-hierarchy FROM entity-type TO entity-type

Figure 4. Generic meta model of the object-role approach

4. EER GENERIC META MODEL *VS.* ORM GENERIC META MODEL

Table 1 details the Atzeni and Torlone (1991) meta-constructs and the corresponding constructs used by the IEF and the InfoDesigner.

Table 1.Comparison of meta model constructs in generic meta models of ORM & EER.

Meta construct	IEF Construct	InfoDesigner Construct
Lexical	Domain	Label type
	Constraint code	Kind of Constraint-code
Abstract	Entity type	Entity type
Aggregation	Relationship on Entity type	Fact-type on Entity type
Function	Relationship-membership on Entity-type and Relationship-type	Role on Entity-type and Fact-type
	Constraint FROM Relationship-membership TO Constraint-code	Constraint FROM Role TO Kind-of-constraint
	Attribute FROM Entity-type TO Domain	Sometimes entity reference mode FROM label-type TO entity type; sometimes a fact type on entity type.
	Identifier FROM Attribute, Relationship-membership TO Entity-type	Entity-identification FROM Label-type TO Entity-type.
	Super-type-partition FROM Entity type TO attribute	Type-hierarchy FROM entity-type TO entity-type
	Super-type-partition-value FROM Super-type-partition, Domain TO Entity-type	-

It can be seen that although many of the concepts have different names, a significant degree of similarity exists between them. Entity types are abstract concepts in both approaches, and an aggregation of entity types is also supported in both approaches. Both approaches have two lexical constructs, one representing the allowable values for instances of data, and the other representing the allowable constraint codes that may be defined. More kinds of constraint are supported by ORM than are supported by EER, although these kinds are not detailed in the generic meta models shown.

A number of the functions defined for both models, such as those used to define roles and relationship-memberships, entity-identification, attributes, constraints, partitions and type-hierarchies are very similar. The IEF has additional functions to support the definition of the partitioning attribute in a super-type entity, and the association of a partitioning attribute value with each of the sub-entity types of the type hierarchy.

5. MAPPING BETWEEN THE EER AND ORM GENERIC META MODELS

Although many of the concepts in both meta models appear similar, great care needs to be taken when defining mapping algorithms from one meta model to the other due to problems of synonymy, homonymy and polysemy. Tables 2 and 3 represent the suggested algorithm for the two way mapping between EER constructs and ORM constructs.

Table 2. Mapping EER to InfoDesigner ORM

Meta construct	EER Construct	ORM Construct
Lexical	Domain	Becomes an ORM label type or an ORM entity type plus an ORM label type depending on the application semantics. Where the domain supports a number of different EER attribute types it is more likely to become an ORM entity type plus label type. The label type will provide a reference mode to the ORM entity type.
	Constraint code	Becomes Kind-of-constraint-code. Both approaches support constraint types not supported by the other. Some constraints will therefore need to be supported procedurally.
Abstract	Entity type	Becomes ORM Entity type.
Aggregation	Relationship type	Becomes ORM Fact-type.
Function	Relationship-membership	Becomes ORM Role.
	Attribute / Identifier	Becomes either a fact type or a reference mode depending on application semantics. Where appropriate the EER attribute becomes a fact type between the ORM entity type corresponding to the EER entity type which the attribute is describing, and the ORM entity type representing the EER domain on which the attribute is based. Where appropriate the EER attribute will become a reference mode between the ORM entity type corresponding to the EER entity type which the attribute is describing, and the label type corresponding to the domain on which it is based. Each EER attribute forming part of the EER entity identifier establishes the corresponding ORM label type as part of the ORM entity identifier. Where an EER relationship membership forms part of the identifier, the steps outlined above for attributes apply to the identifying attributes of the related EER entity type.

Table 2. (continued)

Meta construct	EER Construct	ORM Construct
	Constraint	Becomes ORM constraint subject to note above.
	Super-type-partition	Becomes type-hierarchy
	Super-type-partition-value and entity-type pair	The super type ORM entity type plays a mandatory role in a fact type which relates it to the entity type used to partition into sub types. The association of a value with a sub entity type must be performed procedurally.

Table 3. Mapping ORM to EER

Meta construct	ORM Construct	EER Construct
Lexical	Label	Becomes domain
	Constraint code	Becomes constraint code. Both approaches support constraint types not supported by the other. Some constraints will need to be supported through application models and procedural logic.
Abstract	Entity type	Becomes entity type where at least one role with a simple constraint emanates from it. If not the entity type will be merged with its referencing label type to become an EER domain.
Aggregation	Fact-type	Map unary fact types as Boolean attributes of the relevant entity type. For binaries the fact type will become a relationship where both ORM entity types become EER entity types, otherwise the fact type will become an attribute.
Function	Role	Becomes a relationship membership when its ORM entity type becomes an EER entity type.
	Constraint	Becomes Constraint subject to note above
	Reference mode	Becomes an attribute based on the EER domain corresponding to the ORM label type which is referencing the ORM entity type
	Identifier	Becomes an attribute which is contributing to the identification of the EER entity type that it is describing.
	Type-hierarchy	The super type entity type plays a mandatory role to the entity type used to partition into sub types. The association of a value with an entity sub type must be done procedurally.

6. CONCLUSION

This paper has attempted to develop a framework for conceptual exchange between object-role modelling and extended entity-relationship modelling. Meta models have been identified as a useful means of formalising the description of each modelling technique, and simplified meta models have been developed for ORM and EER modelling techniques. However, expressing each meta model using its own modelling technique does not go far enough in simplifying the conceptual exchange process. While the diagrams associated with both modelling techniques are useful due to their simplicity, clarity and visual representation, they are of limited value in establishing interoperability. When both data modelling techniques are described using a common more abstract language, which is not tied to either approach, the similarities between both approaches are highlighted to a much greater extent than when direct comparisons are drawn. The use of a third, more abstract meta modelling language also appears to have greater potential to simplify the conceptual exchange between a wider variety of data modelling techniques than just the two considered in this paper, although this aspect has not been explored.

The use of meta models in establishing interoperability is useful because it clearly separates the modelling technique components or concepts from their graphical representation. When comparing modelling techniques it is not appropriate to compare only the diagrams, because such a comparison will favour those approaches with richer graphic notations. In many EER approaches, diagrams and the corresponding dictionary entries must be considered as an integrated whole when developing meta models.

The comparisons in this paper provide a useful insight into the similarities and differences between both approaches, and in a number of areas further research is required to extend the transformation algorithms. While some constraints representable in one modelling technique are not representable in the other, when each technique is used in conjunction with other modelling techniques used to represent these aspects, this is not necessarily a weakness. This aspect, involving the use of integrated sets of modelling techniques also requires further research.

This work forms part of the DROTIS project (Steele & Zaslavsky, 1993a) which investigates the problems associated with system modelling techniques integration and interoperability. The goal of DROTIS is to express a given model using different combinations of different modelling techniques and automatically translate between them depending on the preference of the system developer or the particular characteristics of the problem domain. The authors firmly believe that while this problem is currently being investigated mainly in research laboratories, in the near future industry will face significant problems caused by CASE tool incompatibility and difficulties in converting "legacy" software specifications into a form where they can be used by the sophisticated next generation of CASE tools.

7. ACKNOWLEDGEMENT

The authors are thankful to the Faculty of Computing and Information Technology at Monash University, Melbourne, Australia for providing financial support for this project.

8. REFERENCES

ANSI (1992), "ANSI Draft IRDS Conceptual Schema", X3H4.6/92-001R3, X3H4/92-003R3 American National Standards Institute.

Atzeni, P. & Torlone, R. *A* (1991), Meta model approach for the management of multiple models in CASE-tools, *in:* "Database and Expert Systems Applications", Proceedings of the International Conference in Berlin, Federal Republic of Germany, 1991 / D. Karagiannis, ed, Springer-Verlag, 1991, pp.350-355.

Beynon-Davies, P., (1992), Entity models to object models: object oriented analysis and database design, *Information and Software Technology*, Vol.34, No 4, April 1992, pp.255-262.

ECMA, (1990), European Computer Manufacturers Association, Portable Common Tool Environment Standard, ECMA- 149, European Computer Manufacturers Association.

EIA, (1991), Electronics Industries Association, CASE Data Interchange Format, Interim Standards, July, 1991.

Halpin, T. & Oei, J.L.H., (1992), A framework for comparing conceptual modelling languages, Technical report No.92-29, Department of Informatics, Katholieke Universiteit Nijmegen, November.

Imber, M., (1991), CASE data interchange format standards, *Information and Software Technology*, Vol.33, No.9, November, 1991, pp.647-655.

Laender, A. & Flynn, D., (1993), A semantic comparison of the modelling capabilities of the ER and NIAM models, *in:* Proceedings of the Twelfth International Conference on the Entity Relationship Approach, Dallas, Texas, December 15-17, p 237-252.

Martin, J. & McClure, C., (1985) "Diagramming Techniques for Analysts and Programmers", Prentice-Hall, New-York.

Nijssen, G.M & Halpin, T.A., (1989) "Conceptual Schema and Relational Database Design", Prentice Hall, Sydney.

Oei, J.L.J., van Hemmen, L.J.G.T., Falkenberg, E.D. and Brinkkemper, S., (1992), The meta model hierarchy: a framework for information systems concepts and techniques, Technical report 92-17, Department of Information Systems, University of Nijmegen.

Steele, P.M. & Zaslavsky, A.B., (1993a), CASE tool support for integration of system modelling techniques, *in:* Proceedings of the Fourth Australian Conference on Information Systems, Brisbane, QLD, September, 1993.

Steele, P.M. & Zaslavsky, A.B., (1993b) The role of meta models in federating system modelling techniques, *in:* Proceedings of the 12th International Conference on the Entity Relationship Approach, Dallas, Texas, December 15-17, p 301-312.

USING AN EXPLICIT SYSTEM OF CONCEPTS FOR INFORMATION SYSTEMS MODELLING: COMOD

João Alvaro Carvalho and Luis Amaral

Universidade do Minho, Departamento de Informática, 4719 BRAGA CODEX, PORTUGAL

Abstract

Information systems models are central to the process of information systems development. Information systems development methods provide guidelines for modelling information systems. Usually, those guidelines explicate how to use some representation technique to build a model but they seldom are explicit about the system of concepts that underlies the development method.

This paper presents a definition of *system of concepts* for information systems modelling, describes the knowledge that a system of concepts should include, and proposes a system of concepts - COMOD - to be used to build information systems models.
COMOD is a system of concepts with roots on conventional approaches to information systems development. Although it provides a basis for information systems modelling, COMOD is not a development method as it it doesn't contain any guidelines for conducting the modelling or the development processes.

1. INTRODUCTION

Recent research in the area of information systems led to an approach to the study of information systems development that stresses the importance of understanding information systems methods (meta-modelling) (Wijers 1991), (Brinkkemper 1991) in order to 'design' methods which are adequate to specific situations (method engineering).

One of the aspects of meta-modelling is to understand information systems models. This demands detailed knowledge about the 'constructs' used in the models and how they interrelate. This paper defines 'system of concepts' as a set of modelling concepts (or constructs), their associations and a set of rules that affect the concepts and their associations. It points out that the systems of concepts of information systems development approaches are seldom articulated what contributes to a lack of integration of the partial models that constitute a full information system model.

The paper presents a system of concepts - COMOD (Carvalho 1991, 1992) - that has been built by 'merging' the modelling concepts of several information systems development methods. COMOD is an attempt to integrate partial models, to provide a sound basis for model validation and constitutes a general framework for information systems modelling.

This paper is structured as follows: Section 2 refers to the roles played by information systems models in the information systems development process and highlights the importance of expliciting the modelling concepts that underlie information systems models. The concept of system of concepts is used to refer to the set of modelling concepts their associations and rules that affect them used in an information systems development approach. Section 3 enumerates the components that should be included in a system of concepts. Section 4 briefly describes COMOD, a system of concepts for information systems modelling that has its roots on 'conventional' approaches to information systems development. Section 5 describes the way COMOD has been built and evaluated.

2. INFORMATION SYSTEMS DEVELOPMENT AND INFORMATION SYSTEMS MODELS

Information systems development can be described as an activity that leads to the improvement of an information system by means of the creation of computer-based (information) systems that provide support to the collection, storage, processing and/or distribution of information.

During the information systems development process, information systems technicians have to deal with two different 'objects' that correspond to two different kinds of information systems: (i) the human activity systems that gather, process, store and deliver information that is necessary to carry out organizational activities; (ii) computer-based information systems that support (i).

Typically, the process of information systems development starts with information systems technicians trying to understand an organizational information system and creating models of it. These are descriptive models that depict what the information system is and what it is supposed to do. Later on, prescriptive models for the computer-based information systems to be built are produced: first, a statement of the requirements for the computer-based information system; then, a detailed model of that system.

These models play several 'roles' across the development process, namely:

(i) to support the communication among the members of the development team in order to ensure that they have a common understanding of the system being studied;

(ii) to support the communication between the development team and the 'users' so the latter can validate the developers understanding of the system;

(iii) to keep record of the design decisions made during the development process.

It is easy to recognize that information systems models are central to the process of information systems development. Therefore, to improve the way information systems are developed it is important to study information systems models and the process of information systems modelling. The ideas presented in this paper are generally applicable to all types of models used in the information systems development process. However, the paper is specially concerned with the models built during the early stages of the development process (information systems analysis) - the descriptive models that depict the information system as a human activity system and describe what it is and what it is supposed to do.

When looking to an information system, developers try to identify system components that are expected according to some ontological position they assume. For example, if developers are using a 'conventional' approach they will look for things like: processes (something or someone that manipulates data); files (data repositories); flows (data 'traveling' in the system); data items; entities (aggregations of data items); relationships (associations) between entities; events (happenings to which the system has to react); states and state transitions; triggers (whatever activates processes); and so on. If developers adopt an 'object-oriented' approach they will be looking for things such as: classes and objects; properties; messages between objects or objects and classes; methods or services that manipulate the properties of an object or a class; etc..

Some of these 'things' or 'constructs' are easy to identify as they have a direct correspondence to real world components (e.g., a file). Others demand some capacity of abstraction (e.g., entity, object) and might be more difficult to find out during a development process.

The 'things' (or 'constructs') enumerated above may be described as *modelling concepts* used by developers to model an information system. In this paper, the set of modelling concepts used to model an information system, the associations among them and a set of rules that affect them is referred to as a *system of concepts*.

Different information systems development methods suggest different systems of concepts. Sometimes the differences are minor. For example, the systems of concepts underlying methods such as Structured Analysis and SSADM have many things in common.

However, the differences between Structured Analysis or SSADM and Coad & Yourdon object-oriented method are easy to identify.

It might be interesting to note that, although a system of concepts is inherent to any development method, it is seldom made explicit by methods. To identify the system of concepts of a method, one has to go through the method textbooks, paying special attention to the representation techniques suggested by the method, identifying the concepts being used and inferring their associations and rules. Moreover, there are associations between modelling concepts or rules that affect them which are never made explicit. The lack of such an explicit system of concepts leads to the inexistence of a comprehensive and integrated view of an information system. A model of an information system is a often a collection of sub-models placed side by side instead of a set of inter-related sub-models. Furthermore, the system of concepts is the basis for several inference mechanisms related to the validation of the information systems model.

3. COMPOSITION OF A SYSTEM OF CONCEPTS

In order to fully describe a system of concepts several aspects have to be considered. First of all it is necessary to describe the terms to be used and their meaning (a lexicon).

Moreover, the following components should be considered (Carvalho 1991):

Structure: a description of the way modelling concepts are associated and the roles played by the modelling concepts in the associations. The structure is further characterized by constraints such as: uniqueness constraints; cardinality of the involvement of a modelling concept in an association; compulsiveness of the involvement of a modelling concept in an association.

Derivation rules: rules for deriving model components based on already existing parts of an information system model.

Validity constraints: constraints that are concerned with false knowledge that must not be part of an information system model. They are of two types: value constraints and consistency constraints (used to pursue that there are no contradictory statements in the information system model).

Completeness conditions: rules that provide means to detect missing knowledge in an information system model.

Clearness guidelines: rules that help the identification of situations in a model that are valid but can lead to misunderstanding (e.g., the use of homonyms).

Dependency rules: dependency rules express the dependencies between modelling concepts and associations which are relevant to assess the impact of the change on a information system model.

Heuristics might also be used to complement the type of rules described above.

4. COMOD (CONCEPTS FOR INFORMATION SYSTEMS MODELLING)

COMOD is a method-independent system of concepts that includes all the components referred to in the previous section.

COMOD includes and integrates modelling concepts used in two levels of information system models:

(i) general models of the global information system of an organisation (e.g., the information system architecture) and the descriptions of the information needs of several organisation components (e.g., organisational units, function, managers);

(ii) detailed descriptions of information (sub-)systems covering the three perspectives of information systems (data, processes and behaviour[1]).

In spite of its method-independence, COMOD is based on 'conventional' methods such as BSP, SA, SSADM, IE.

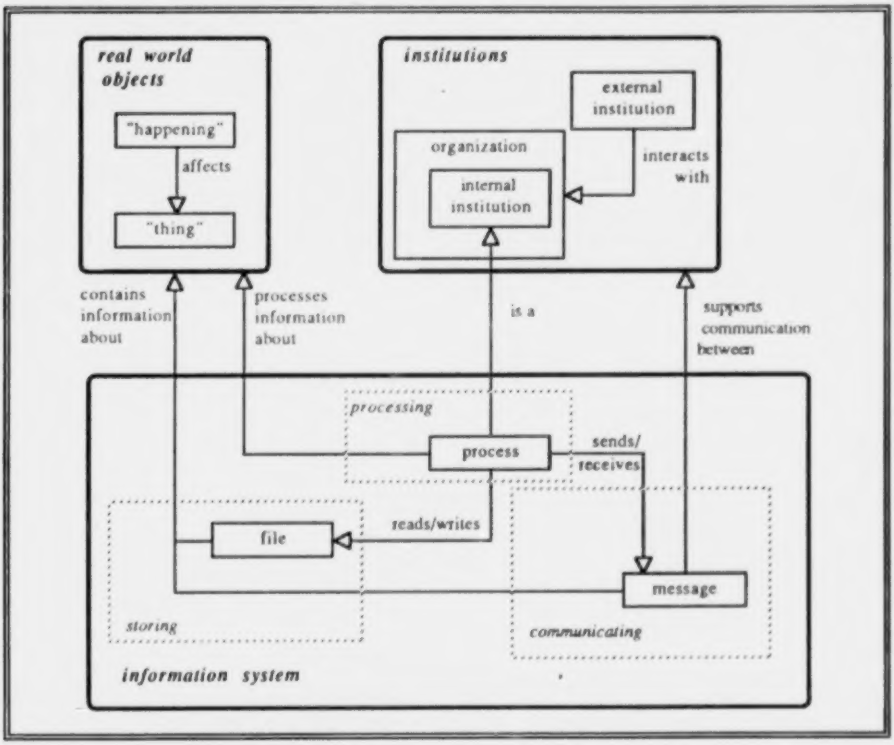

real world objects
"happening"
affects
"thing"
institutions
external institution
organization
internal institution
interacts with
contains information about
processes information about
is a
supports communication between
processing
process
sends/receives
file
reads/writes
message
storing
communicating
information system

Figure 1 - The three main components of COMOD and their relationships

COMOD attempts to provide a framework to view information systems in an integrated manner. A general view of COMOD is presented in figure 1. Modelling concepts are arranged in three groups: (a) institutions - organisations or components of an organisation's structure; (b) real world objects - relevant 'things' and 'happenings' in the problem domain; and (c) the information system as the system that communicates, stores and processes information in an organisation (Buckingham et al. 1987). Figures 2, 3 and 4 further describe each of these three COMOD components.

---

[1] These three perspectives are widely recognized in the information systems community [Olle et al. 1988], [Nijssen, Halpin 1989]. Nijssen [1989] refers to these three perspectives as the *how* the *what* and the *when*. However, it might be better to say: *what* happens, *to what* (whom?) it happens and *when* it happens. Processes and data perspectives address the *static* aspects of an information system while the behaviour perspective addresses the *dynamic* aspects.

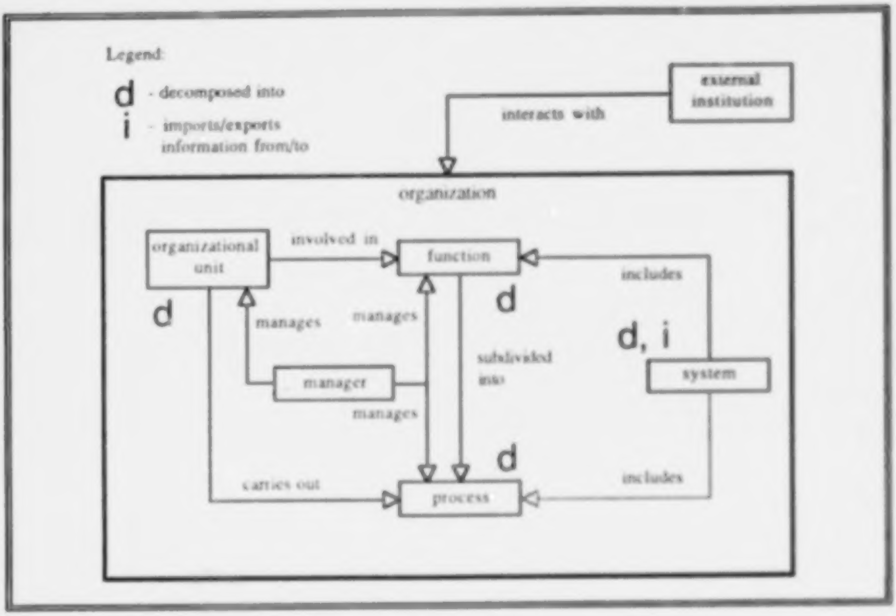

Legend:
d - decomposed into
i - imports/exports information from/to
external institution
interacts with
organization
organizational unit
involved in
function
includes
d
d
manages
manages
d, i
subdivided into
manager
system
manages
d
carries out
process
includes

Figure 2 - The institutions

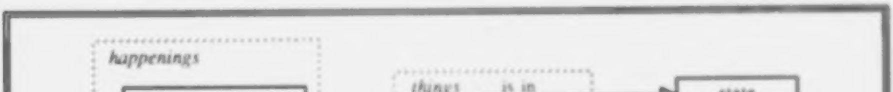

happenings
internal event
s
event
affects
things
is in
state
s
entity
has
r
cardinality
external happening (incident)
key
perceived through the reception of
is an aggregation of
message
conveys values of
data item
is a set of
has
Legend:
s - there is a sequence of
r - has a relationship with
value type
derivation rule
value constraint

Figure 3 - The real world objects

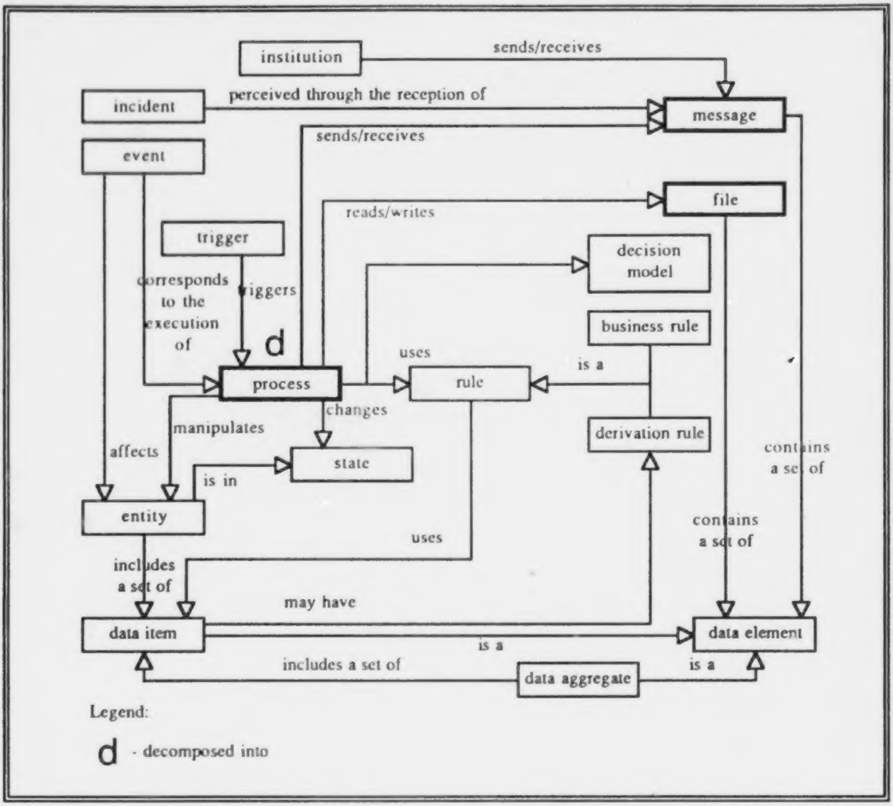

institution
sends/receives
incident
perceived through the reception of
message
sends/receives
event
file
reads/writes
trigger
decision model
corresponds to the execution of
triggers
d
business rule
uses
is a
process
rule
changes
derivation rule
manipulates
affects
state
contains a set of
is in
entity
contains a set of
uses
includes a set of
may have
data item
data element
is a
includes a set of
data aggregate
is a
Legend:
d - decomposed into

Figure 4 - A detailed vision of the "information system" perspective

It is worth noting that high-level models of information systems combine detailed representations of institutions with raw representations of real world 'things' (groups *a* and *b* of the previous paragraph). The objective is to characterize the information needs of organisation components. This view is illustrated in figure 5.

Detailed models of information (sub-)systems are built by developing the views based upon modelling concepts of groups *b* and *c* of the previous paragraph.

Figures 1 to 5 constitute an informal presentation of COMOD. A complete and rigorous description using NIAM's formalism for conceptual data modelling has been made for representing COMOD's structure. The NIAM model was complemented by a set of rules (An extract of this COMOD description is presented in the appendix).

A *prolog* description of COMOD was used to build a prototype of a repository for information system models with advanced validation facilities.

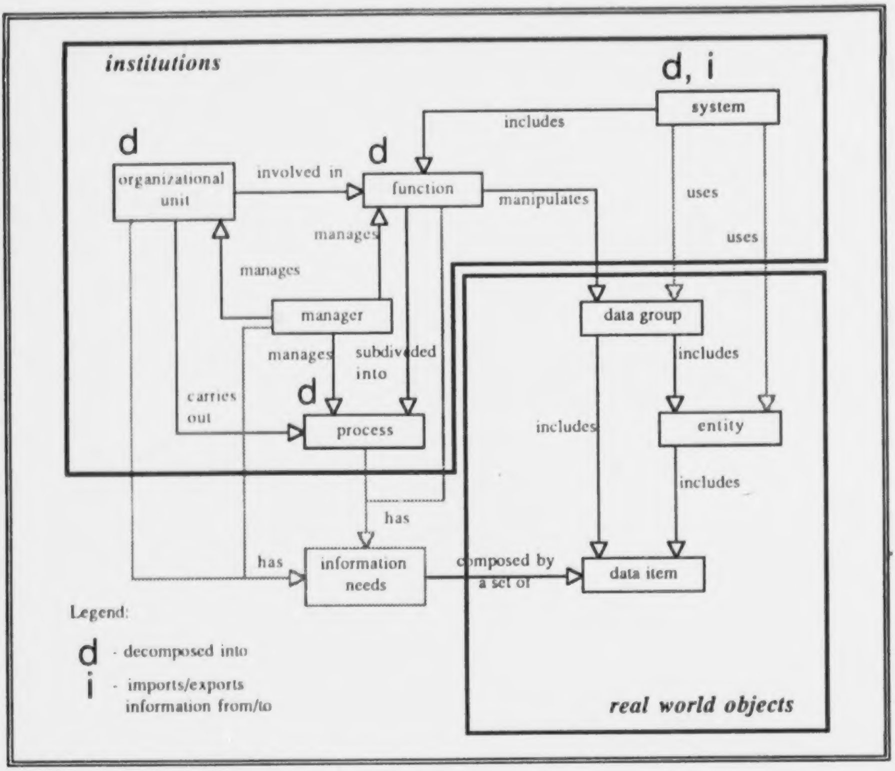

institutions
d, i
system
includes
d
d
organizational unit
involved in
function
manipulates
uses
uses
manages
manages
manages
manager
data group
manages
subdivided into
includes
carries out
d
process
includes
entity
includes
has
has
information needs
composed by a set of
data item
Legend:
d - decomposed into
i - imports/exports information from/to
real world objects

Figure 5- The organisation's information requirements and the organisation's information system architecture

5. COMOD CONSTRUCTION AND EVALUATION

The construction of COMOD was an iterative process that combined analysis of:

a) information systems development methods
(e.g., **BSP** (IBM 1984), **CTM** (ter Hofstede, Brinkkemper 1989), **ERM** (Chen 1976), **INFOLOG** (Fiadeiro, Sernadas 1986), (Sernadas 1982), (Sernadas, Sernadas 1983, 1986), **IE** (Finkelstein 1989), (MacDonald 1986), (Martin 1986a, 1986b, 1987), **JSD** (Cameron 1986, 1989), (Sutcliffe 1988), (Jackson 1983), **MERISE** (Collongues et al. 1986), **MULTIVIEW** (Avison, Wood-Harper 1990), (Wood-Harper et al. 1985), **NIAM** (Nijssen, Halpin 1989), **RUBRIC** (Loucopoulos, Layzell 1989), (RUBRIC 1989), **SA** (DeMarco 1979), (Gane, Sarson 1979), **SSADM** (Ashworth, Goodland 1990), (Downs et al. 1988), (Longworth, Nicholls 1986), (Nicholls 1987));

b) review of comparative studies on development methods
(e.g., CRIS reports (Olle et al. 1982, 1983, 1986); AMADEUS project reports (Black et al. 1987), (AMADEUS 1986a); (Blank, Krijger 1983), (Maddison et al. 1983), (Silva 1985));

c) comparison with other systems of concepts (although none was as comprehensive as COMOD): AMADEUS' meta 'unified model' (Black et al.1987), (AMADEUS 1986b); IFIP's framework for methods understanding (Olle et al. 1988); BSP's data model (Sakamoto, Ball 1982); ISO's concepts and terminology (van Griethuysen 1982); NIAM's conceptual meta schema (Nijssen, Halpin 1989); BPSM's data model for data administration (BPSM 1987).

The COMOD construction process can also be viewed as the result of combining two efforts with opposite 'strategies': (i) an attempt to synthesize/integrate the modelling approaches of several development methods (a bottom up process); (ii) and an attempt to provide a framework for understanding information systems in organisations (a top down process).

COMOD evaluation was done by applying it to two different cases. One was a real case based on a project where the authors have been involved as consultants. The project included the determination of an organisation's information system architecture and the definition of the design requirements for a computer-based system to support one of the organisation's sub-systems. The other case was the specification of the design requirements for the repository system to support COMOD itself (Carvalho 1991). Both cases have been used to test that repository system.

6. CONCLUSION

Information systems modelling (in the context of information systems planning and development) implies an 'ontological position' about information systems i.e., a set of assumptions on what an information system is and how it should be described. Such 'ontological position' may be expressed as a system of modelling concepts.

A system of concepts is more than a list of concepts as it should include a set of associations among the concepts and a set of rules that affect both concepts and their associations.

This paper briefly describes COMOD (**CO**ncepts for information systems **MOD**elling) which is a system of concepts with roots on conventional approaches to information systems development although some of the concepts used in COMOD differ from those of widely divulged methods such as SSADM, SA or IE.

It should be stressed that COMOD is not a method for information systems development. COMOD provides the basis for information systems modelling but doesn't contain any guidelines for conducting the development process or for building the information systems models.

A parallel line of work is now being pursued to develop a system of concepts for supporting object-oriented approaches to information systems modelling. The comparison of both systems of concepts can contribute to a better understanding of the information systems development process.

7. ACKNOWLEDGMENT

The research work has been supported by Fundação Calouste Gulbenkian (Portugal) under grant number 13/88/B.

8. REFERENCES

AMADEUS, (1986), AMADEUS Project: Deliverable D1, UMIST.
AMADEUS, (1986), AMADEUS Project: Deliverable D3, UMIST.
Ashworth, C. and Goodland, M., (1990), "SSADM: A Practical Approach", McGraw-Hill.

Avison, D.E. and Wood-Harper, A.T., (1990), "MULTIVIEW: An Exploration in Information Systems Development", Blackwell.

Black, W., Layzell, P., Loucopoulos, P. and Sutcliffe, A., (1987), "AMADEUS project: Final Report", UMIST, ISRG-106.

Blank, J. and Krijger, M., eds., (1983), "Software Engineering: Methods and Techniques (Evaluation of Methods and Techniques for the Analysis, Design and Implementation of Information Systems)", John Wiley & Sons.

BPSM, (1987), "BPSM: Sistema de Dicionário de Dados", BPSM - Banco Pinto & Sotto Mayor, Direcção de Organização e Informática (DOI), Lisboa.

Brinkkemper, S., (1990), "Formalisation of Information Systems Modelling", Thesis Publishers, Amsterdam.

Buckingham, R., Hirschheim, R., Land, F. and Tully, C., (1987), Information systems curriculum: a basis for course design, *in*: "Information Systems Education: Recommendations and Implementation", Buckingham, R., Hirschheim, R., Land, F. and Tully, C., eds., Cambridge University Press.

Cameron, J.R., (1989), An overview of JSD, *IEEE Transactions on Software Engineering*, Vol. 12, pp. 222-240.

Cameron, J.R., (1989), "JSP and JSD: The Jackson Approach to Software Development", IEEE Computer Society Press, (2nd edition).

Carvalho, J.A., (1991), "BMKB (Business Meta Knowledge Base): A Repository of Models for Assisting the Management and Development of Organizational Information Systems", Ph.D. Thesis, UMIST, U.K..

Carvalho, J.A., (1992), "COMOD: A System of Concepts for Information Systems Modelling", Universidade do Minho, Technical Report.

Chen, P., (1976), The entity-relationship model: towards an unified view of data, *ACM Transactions on Database Systems*, Vol. 1, 1.

Collongues, A., Hugues, J. and Laroche, B., (1986), "MERISE: Méthode de Conception", Dunod.

DeMarco, T., (1979), "Structured Analysis and Systems Specification", Yourdon Press.

Downs, E., Clare, P. and Coe, I., (1988), "SSADM - Structured Systems Analysis and Design Method: Application and Context", Prentice Hall.

Fiadeiro, J. and Sernadas, A., (1986), The INFOLOG linear tense propositional logic of events and transactions, *Information Systems*, Vol. 11, 1, pp. 61-85.

Finkelstein, C., (1989), "An Introduction to Information Engineering: From Strategic Planning to Information Systems", Addison-Welsley.

Gane, C. and Sarson, T., (1979), "Structured Systems Analysis: Tools and Techniques", Prentice-Hall.

IBM, (1984), , Business Systems Planning - Information Systems Planning Guide", IBM, (4th edition).

Jackson, M., (1983), "System Development", Prentice-Hall.

Longworth, G. and Nicholls, D., (1986), "SSADM Manual", (Version 3), NCC Publications, Manchester.

Loucopoulos, P. and Layzell, P., (1989), Improving information systems development and evolution using a rule-based paradigm, *Software Engineering Journal*, Vol. 4, 5, pp. 259-267.

MacDonald, I., (1986), Information engineering: an improved, automatable methodology for the design of data sharing systems, *in*: "Information Systems Design Methodologies: Improving the Practice", Olle, T., Sol, H., Verrijn-Stuart, A., eds., North-Holland, Amsterdam, pp. 173-224.

Maddison, R., Baker, G., Bhabuta, L., Fitzgerald, G., Hindle, K., Song, J., Stokes, N. and Wood, J., (1983), "Information System Methodologies", Wiley Heyden.

Martin, J., (1986), "Information Engineering (Volume 1)", SAVANT, Carnforth.

Martin, J., (1986), "Information Engineering (Volume 2)", SAVANT, Carnforth.

Martin, J., (1987), "Information Engineering (Volume 3)", SAVANT, Carnforth.

Nicholls, D., (1987), "Introducing SSADM: The NCC Guide", NCC Publications, Manchester.

Nijssen, G., (1989), "A Evaluation of the IFIP ISM88 Data Structure Diagram Formalism", IFIP WG 8.1 CRIS Review Workshop.

Nijssen, G. and Halpin, T., (1989), "Conceptual Schema and Relational Database Design", Prentice-Hall.

Olle, T., Verrijn-Stuart, A. and Bhabuta, L., eds., (1988), "Computerized Assistance During the Information Systems Life Cycle", North-Holland, Amsterdam.

Olle, T., Sol, H. and Verrijn-Stuart, A., eds., (1982), "Information Systems Design Methodologies: a Comparative Review", North-Holland, Amsterdam.

Olle, T., Sol, H. and Tully, C., eds., (1983), "Information Systems Desigm Methodologies: a Feature Analysis", North-Holland, Amsterdam.

Olle, T., Sol, H. and Verrijn-Stuart, A., eds., (1986), "Information Systems Design Methodologies: Improving the Practice", North-Holland, Amsterdam.

Olle, T., Hagelstein, J., Macdonald, I., Rolland, C., Sol, H., Van Assche, F. and Verrijn-Stuart, A., (1991), "Information systems Methodologies: A Framework for Understanding", 2nd edition, Addison-Wesley.

RUBRIC (1989), "RUBRIC Concepts Manual (version 5)", ESPRIT Project 928.

Sakamoto, J. and Ball, F., (1982), Supporting business systems planning studies with the DB/DC data dictionary, *IBM Systems Journal*, Vol. 21, 1, pp. 54-80.

Sernadas, A. and Sernadas, C., (1986), Infolog 86, *Revista de Informática*, Vol. 5, 10, pp. 9-17.

Sernadas, A., (1982), Information systems specification with infolog, *in*: "Evolutionary Information Systems", Hawgood, J., ed., North-Holland, Amsterdam.

Sernadas, C. and Sernadas, A., (1983), "Introdução à Metodologia Infolog", Infolog RR04, Faculdade de Ciências de Lisboa.

Silva, L., (1985), "Metodologias de Desenvolvimento de Sistemas Informáticos", Dissertação de Mestrado, Universidade do Minho, Braga.

Sutcliffe, A., (1988), "Jackson System Development", Prentice Hall.

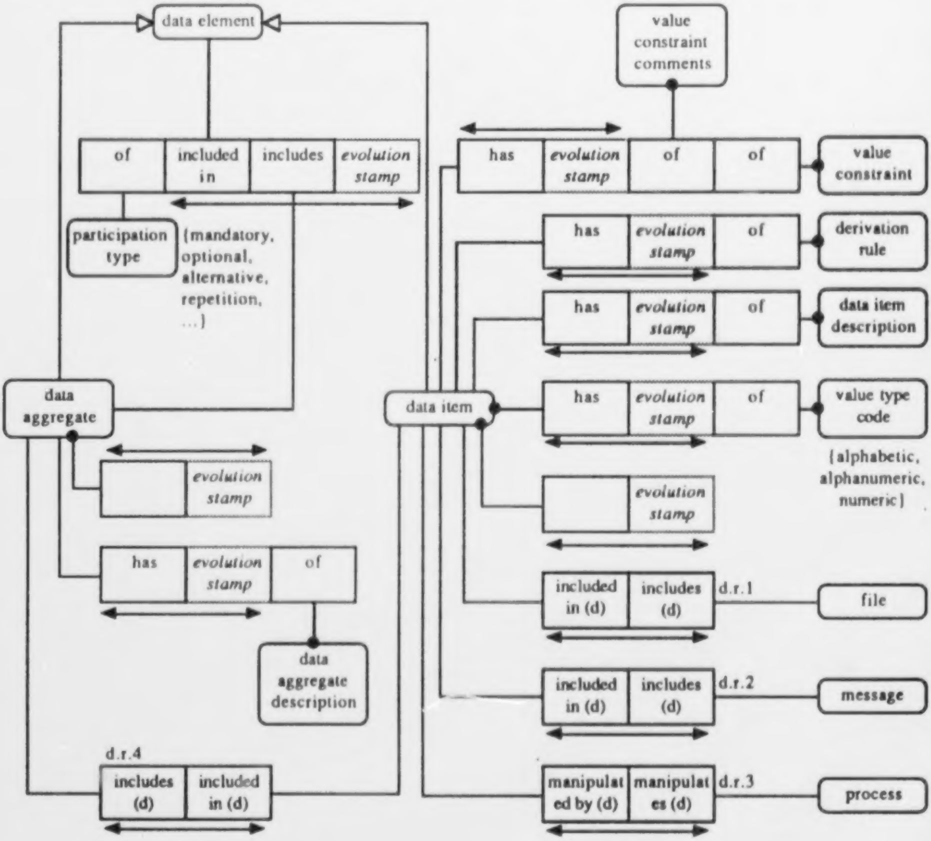
ter Hofstede, A. and Brinkkemper, S., (1989), "Conceptual Task Modelling", Internal Report 89-14, University of Nijmegen, Faculty of Mathematics and Informatics.

van Griethuysen, J., ed., (1982), "Concepts and Terminology for the Conceptual Schema and Information Base", Report ISO TC97/SCS/WG3.

Wijers, G., (1991), "Modelling Support in Information Systems Development", Thesis Publishers.

Wood-Harper, A.T., Antill, L. and Avison, D., (1985), "Information Systems Definition: The Multiview Approach", Blackwell Scientific Publications.

APPENDIX - EXTRACT FROM THE FORMAL DESCRIPTION OF COMOD


data element
value constraint comments
of
included in
includes
evolution stamp
has
evolution stamp
of
of
value constraint
participation type
{mandatory, optional, alternative, repetition, ...}
has
evolution stamp
of
derivation rule
has
evolution stamp
of
data item description
data aggregate
data item
has
evolution stamp
of
value type code
{alphabetic, alphanumeric, numeric}
evolution stamp
evolution stamp
has
evolution stamp
of
data aggregate description
included in (d)
includes (d)
d.r.1
file
included in (d)
includes (d)
d.r.2
message
d.r.4
includes (d)
included in (d)
manipulated by (d)
manipulates (d)
d.r.3
process

d.r.1: A *file* F **includes (d)** *data item* DI if:

◊ F **includes** *data element* DI

or

◊ There is a *data aggregate* DA such that:

(F **includes** *data element* DA) and (DA **includes (d)** *data element* DI).

d.r.2: A *message* M **includes (d)** *data item* DI if:

◊ M **includes** *data element* DI

or

◊ There is a *data aggregate* DA such that:

(M **includes** *data element* DA) and (DA **includes (d)** *data element* DI).

d.r.3: A *process* P **manipulates** *data item* DI if:

◊ There is a *file* F such that:

(DI **is included in** F) and (F **is read by** P)

or

◊ There is a *file* F such that

(DI **is included in** F) and (F **is written by** P)

or

◊ There is a *message* M such that:

(DI **is included in** M) and (M **is received by** P)

or

◊ There is a *message* M such that:

(DI **is included in** M) and (M **is sent by** P).

d.r.4: A data aggregate DA **includes (d)** *data item* DI if:

◊ DA **includes** *data element* DI

or

◊ There is a *data aggregate* DA1 such that:

(DA **includes** *data element* DA1) and (DA1 **includes (d)** *data element* DI).

CONJOINED CO-EVOLUTIVE INCREMENTALISM FOR INFORMATION SYSTEMS DEVELOPMENT

Nagib Callaos and Belkis de Callaos

Universidad Simon Bolivar, Dpto. de Procesos y Sistemas, Apdo. Postal 89000, Caracas, Venezuela

Abstract

Elsewhere [Callaos and Callaos, 1994] we proposed a systemic methodological support for information systems development. One of our conclusions was about the importance of paralleling analysis and synthesis, and its internal detailed steps, which are usually executed serially in the traditional systematic methodologies, as in the case of "waterfall model". Parallelism and concurrence between analysis and synthesis, and among theirs detailed internal steps, make possible the existence of feedback and feedforward loops in order to achieve the flexibility required by the adaptability to a dynamic environment.

But to manage an information system development project, requires milestones and, then, serial activities. Since analysis and synthesis (and theirs detailed internal steps) are not serial anymore in the framework we are proposing, they can support no milestones for project management. Consequently we are proposing a paradigmatic shift for milestones identification, based on a Conjoined Co-Evolutive Incremental Methodology. Instead of serializing analysis and synthesis, we propose to serialize effectiveness and efficiency, and their respective internal detailed activities, which will be given in the paper. We will find these internals activities by crossing fundamental dichotomies, such as thinking/acting, external/internal, feedback/feedforward, etc. As a result, we will have seven activities oriented toward achieving effectiveness, and seven oriented to achieving adequate efficiency.

1. INTRODUCTION

Elsewhere [Callaos and Callaos, 1991] we buckled down to define method, tool, technique and methodology. We concluded then than a method is a thinking and/or acting "way" and, as such, it has initial and final "points" defining it. Instrument is the means that allows us to get "transported" thru a given method. A technique is the capacity required for the adequate use of a given tool and, as such, it comprehend good aptitude and the right attitude. Methodology is a set of related or "relationable" methods with their respective tools and techniques, it is a "network of methods", a graph of ways for thinking and/or acting or doing, in order to achieve a given set of objectives. The notion of "methodology" should not be confused with that of method. A graph is conceptually different from its arcs and its nodes. A network of highways, avenues and streets is not to be confused neither with one of its highways, avenues or streets (or a sequence of them), nor with the cars (i.e. tools) traveling in the network, nor with the driving training (techniques) of the cars' drivers. A Computed Assisted Software Engineering (CASE) is not a methodology, it is a tool, a set or a system of tools. The top-down approach of Systems Analysis, for example, is a method, the Data Flow Diagrams are tools, and the aptitude to elaborate Data Flow Diagram is a technique. A methodology of developing information systems could include different top-down alternate methods for systems analysis, it does not to be necessarily reduced to just one top-down method, as is frequently the case for authors that confuse the concept of methodology with the concept of method. A methodology for developing information systems could even include bottom-up methods accompanying top-down methods and, consequently, complementing each other. This inclusion of top-down and bottom-up methods in the same methodology is seen as contradictory and non-sense by those authors, consultants and practitioners that confuse the concept of "methodology" with "method". But, with the definition and conceptual distinction we have done [Callaos and Callaos, 1991], a methodology could contain opposite methods in order to integrate them in a whole, if a particular situation calls for it. Otherwise, if methodology is confused with "method", the possibility of complementary polar opposition vanishes and a contradictory situation emerges.

2. CONJOINING TOP-DOWN AND BOTTOM-UP METHODS

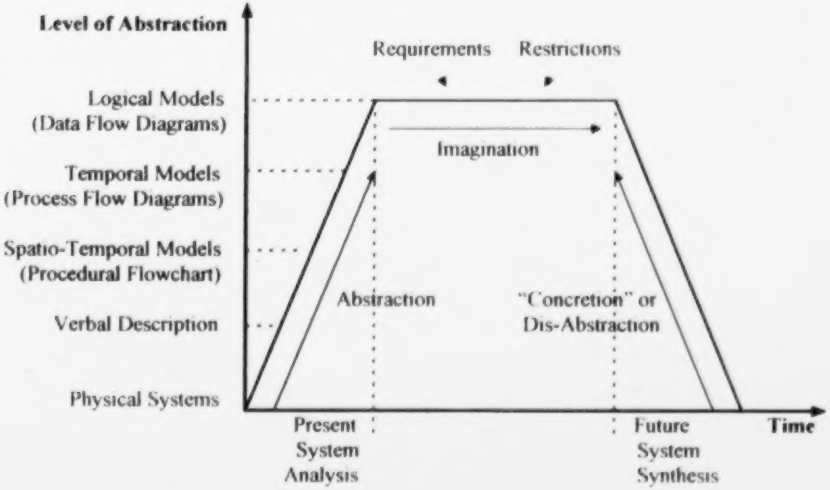
It is a tautology to state that a "top-down" method goes from the "top" to the "down", and a "bottom-up" method goes from the "bottom" to the "top". But, being this so obvious, why in most computer installations or software houses the methodical standard, in the last 20 years, has ben a "top-down" one? How can we apply a "top-down" method if we are in a bottom situation? Shouldn't we apply a "bottom-up" method to get to the "up" and then, and only then, it will be feasible to apply a "top-down" method? Aren't both methods polar opposites and not the contradictory kind of opposites? If so, why most computer installations and software houses exclude the use of "bottom-up" methods when they standardize through the use of "top-down" methods? To use "top-down" methods we should be at the "top" in our initial conditions, and this is not always the case in an information systems development project. A "top-down" method should be used when it is adequate and feasible. If its use is not feasible an alternate method should be used, or a complementary one that would change our initial conditions, as it is the case of the "bottom-up" methods. To use "top-down" methods does not mean necessarily to use them always. And, if for some reason (to reduce maintenance costs, for example) we got to use them always, this doesn't mean that we should use only them, excluding any other or any complementary opposite method.

In the other hand, a "top-down" method could follow the Systems Approach, but this doesn't mean that a systems method is -or got to be- a "top-down" one. Combining "top-down" with "bottom-up" methods is actually a more systemic process, if both methods complement each other or -much better- if they feedback and/or feedforward each other in an adaptive loop. It is more systemic to alternate optional methods, selecting the most suitable one from them, or to combine adequately several of them

[Callaos, 1993]. This point will be retaken later. When we combine both methods in a feedback/feedforward loops, we join them one with other, systemically; i.e. we con-join them. We do not just join them in a set, but we conjoin them in a system, by means of feedback/feedforward loops.

To go from the "top" to the "down" is to go from the "general" to the "specific", from the "abstract" to the "concrete". And to go from the "bottom" to the "up" is the inverse way. So, to use one method or another depends on the level of abstraction the analyst/synthetist is working at. And, since the analysis/synthesis process goes through several levels of abstraction while developing an information systems, then several methods, tools and techniques should be used along the process depending on the level of abstraction at hand.

A general framework for conjoining "top-down" and "bottom-up" methods, tools and techniques, is given in figure 1, where we mapped fundamental levels of abstraction on the time elapsed since the development process began.



Level of Abstraction
Requirements
Restrictions
Logical Models
(Data Flow Diagrams)
Imagination
Temporal Models
(Process Flow Diagrams)
Spatio-Temporal Models
(Procedural Flowchart)
Abstraction
"Concretion" or
Dis-Abstraction
Verbal Description
Physical Systems
Present
System
Analysis
Future
System
Synthesis
Time

Figure 1

The development process of an information system begins usually with an actual (manual and/or computerized) system which effectiveness and/or efficiency need to be improved. This system is a "physical system" and consequently is at a zero abstraction level. Oral and/or written descriptions of a system, or its sub-systems, or its procedures/processes, requires a first level of abstraction, where irrelevant physical aspects are removed, abstracted. A second level of abstraction is achieved when qualitative aspects not related with space and time dimension are abstracted from the verbal description. This second level of abstraction could be represented by spatio-temporal models or diagrams as, for example, procedural flowcharts, with spatial reference, flowcharts with departmental references, etc. A third level of abstraction is achieved when space is abstracted from spatio-temporal models. This third level of abstraction could be represented by temporal models or diagrams, like flowcharts, HIPO diagrams, process flow diagrams, precedence diagrams, etc. A fourth level of abstraction is achieved when we abstract the temporal dimension from the temporal models. In this case the system could be represented by logical models or diagrams, like the very known and used Data Flow Diagrams (DFD). To go from the physical actual system to its logical representation is an abstraction process where bottom-up methods, tools and techniques are very adequate. We suggest not to draw directly the DFD of the actual system as it is proposed by almost all authors in the area of Structured Analysis. It is better, easier and safer to use the "abstraction ladder" shown in figure 1 and described above. The adequacy of our suggestion increases as the complexity of the system increases, and when our access is to detailed information in the lower levels of the organization and not to summarized and general information which usually is provided by the executive level of the organization where the system is, and/or is going to be, inserted.

With the logical models of the actual system, and based on the users requirements and the respective restrictions, the analyst/synthetist forms a mental picture of the proposed future system, by means of his imagination. Then he represents this mental picture by means of models of the highest level of abstraction, i.e., logical models, which could be Data Flow Diagrams. And, from here, an inverse procedure, a "dis-abstracting", a "concretion" process is followed using a "dis-abstracting ladder" analogous to the abstracting one we described above, but used in the inverse way. In this procedure we will go from the logical model of the future system, to its physical implementation, passing through its temporal, spatio-temporal and oral/written modeling. In this way "top-down" and "bottom-up" methods are conjoined systemically and systematically through processes of abstraction, imagination and concretion ("dis-abstraction"). In this way we could insert structured methodologies which are highly systematic into systemic ones. Structured methodologies are highly efficient, but little adaptive, while systemic methodologies are not so efficient, but very adaptive and, hence, more effective [Callaos, 1992]. By inserting systematic methodologies into systemic ones, a good combination of efficiency and effectiveness could be achieved, as we could experience it applying our methodology to approximately 120 projects of information system development.

3. CONJOINING THINKING AND DOING INTO INCREMENTAL CO-EVOLUTION

Elsewhere [Callaos, 1992] we buckled down to the integration of thought and action from a systemic perspective, and concluded that from such a perspective, there's no pre-eminence between action and thinking, between acting and knowing, or between transforming the environment or getting and processing information about it. They are relations of opposite directions but they aren't contradictory, but polar and complementary. Thought does not precede action, linearly, or viceversa. Thought and action relate to each other non-linearly, and synergistically, in a cybernetic loop. Thus, a systemic epistemology (or methodology) should not be centered in the action/transformation relationship, as it is usually suggested in the systems literature [Churchman, 1971, Mattesich, 1978]. Neither should it be centered in the perception/information/knowledge relationship as it was stressed for a long time. It should be centered on both relations simultaneously and on the meta-relation relating them. It should be based on the cybernetic and synergetic loop that relates them in a non-linear way.

The thinking side is based on the conjunction of "top-down" (or deductive) methods with "bottom-up" (inductive) methods. The action side is based on a combination of action-research, action-learning and action-design, which interact with each other through what we can call a "conjoined incrementalism", which is an approach based on the empirical findings of Braybrooke and Lindblom [1970]. We combined their "disjointed incrementalism" with control theory in order to achieve Churchman's pragmatic-teleological truth [Churchman, 1971]. The result was what we called a "conjoined incrementalism" [Callaos, 1994], which we can summarize as follows: (1) according to objectives and means a thinking increment is used to make a general macro-plan and a micro-plan for the next action increment, (2) the micro-plan is executed and feedback loops are applied to achieve the objectives of the action increment, (3) experience drawn from the total execution of the increment is used for feedforward loops in order to adjust the macro-plan and to elaborate the micro-plan of the next step, (4) and so on until achieving the objectives, the "telos", i.e. until achieving Churchman's pragmatic-teleological truth. A string of control loops, alternating feedback, in the action/execution increments, and feedforward in the thinking/planning increments, is the essence of what we called "conjoined incrementalism". In the thinking/planning phases the "bottom-up"-"top-down"-"bottom-up" loop could be applied, in order: (1) to generate the feedforward for the next increment, and (2) to generate general hypothesis for any of the future increments, or new problems or projects. In the action phases information is drawn in order: (1) to generate correcting feedback to achieve the objectives of the increment, and (2) to support the processes of action-research, action-learning and/or action-design.

A prototyping procedure (generating screen prototypes and/or system prototypes) showed to be a very practical support for "conjoined incrementalism", and for a co-evolutionary process through which the system (being designed and implemented) and its environment (users, analysts, etc.) evolve simultaneously by means of affecting each other, reciprocally. In a first thinking increment, a quick "bottom-up" and "top-down" methods (figure 1) are applied to produce a very preliminary screens prototype, which, once perceived and judged by the users, generates feedback and feedforward loops for modifying itself and its respective development project. Then a second "bottom-up" and "top-down" (figure 1) is applied for the generation of a second prototype, and so on until a screen prototype is accepted by the users. At this point a "bottom-up"/"top-down" method is applied to generate a first system prototype which, once implemented, will support an action-learning process from and by the user, by means of its operation and usage. A similar co-evolutive sequence is produced by acting (using the system) and thinking (abstracting, imaging and dis-abstracting or "concreting" by means of "top-down" and "bottom-up" methods). The system acts on the user supporting him in an action-learning process, and the user acts on the system modifying it through new or modified requirements which support the development of the next system prototype, to be done by the analyst whom is inserted in an action-research and action-design.

4. CONJOINING ANALYSIS AND SYNTHESIS

A consequence of conjoining top-down with bottom-up, and thinking with doing, is the necessary conjoinment of analysis and synthesis. In traditional systematic methodologies analysis and synthesis are done sequentially, in series. Likewise the different steps in each one of them, are done in succession, one at a time. The output of each step is the input of the next one. Hence, the name of "waterfall model" that is used frequently to refer to this kind of methodology. But, with the methodology we are briefly describing here, analysis and synthesis, together with its internal step are conducted simultaneously, in parallel, not in series. At any time in the developmental process, analytical and synthetical activities are being done simultaneously. What change over developmental time is the emphasis, the effort, the manpower dedicated to each activity. The developmental process began with high emphasis on analysis, which decreases as the process goes on (figure 2). The design activities began in a lower level of effort than analysis, it increases later and then begin decreasing as the process goes on. The programming (coding and testing) effort has a similar effort pattern, but delayed on relation to the design effort. Likewise with implementation. In this way these activities (and its internal steps) feedback and feedforward each other, providing the necessary (although not sufficient) condition for methodological adaptability.

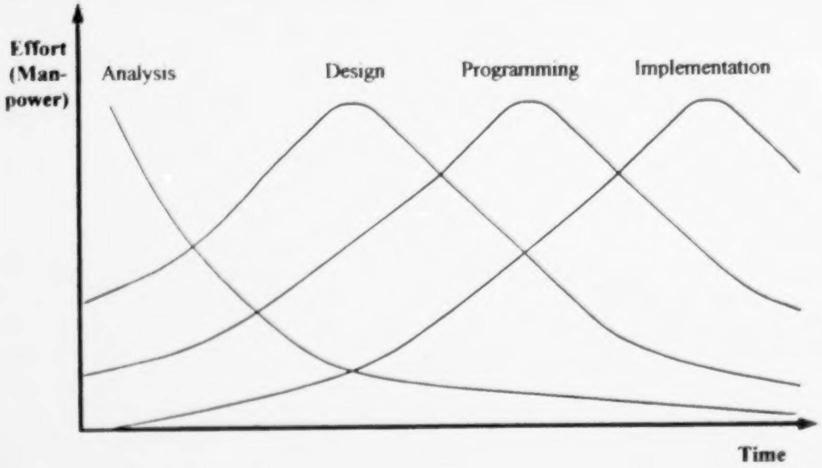


Figure 2

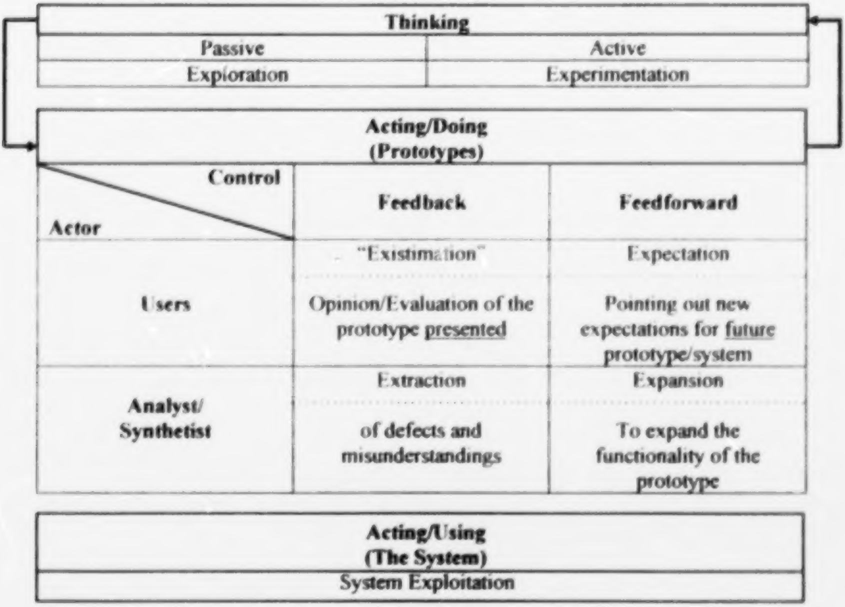
Parallelism provides flexibility in the development process for adaptability to a dynamic environment, and an anticipatory mechanism for dealing with unknown and non-controllable variables, which as such they could have not been foreseen. We can make prevision for a foreseeable event, an event that could happen with some probability. But we cannot make previsions for uncertainty, i.e. for unforeseeable events. In such cases previsions are not feasible and anticipation is very desirable. Parallelism provides means for anticipating undesirable events in order to have enough time into dealing with them. To anticipate or to advance unknown and undesirable events, to bring them forward while there is enough time to the deadline, is a good way -sometimes the only one- for dealing with uncertainty. Parallelism makes possible anticipation for dealing with uncertainty and makes possible feedback and feedforward loops in order to achieve the flexibility required by the adaptability to a dynamic environment [Callaos and Callaos, 1994]

5. EFFECTIVENESS/EFFICIENCY PARADIGM

Traditional methodologies are centered at the analysis/synthesis paradigm. Consequently, information system development project management is based on milestones associated with this paradigm. Analysis milestones first, and synthesis ones later. Likewise, statistics for project estimations are collected according this paradigm. But, if we are to conduct the analysis and synthesis activities simultaneously, the traditional milestones are not useful any more. So, a new paradigm is to be identified for project planning and control. The effectiveness/efficiency paradigm proved to be a good one in planning and controlling more than 120 information system development projects.

Elsewhere [Callaos and Callaos, 1994] we concluded that the so called "software crisis" is an "effectiveness crisis" rather than, or besides, an "efficiency crisis". Then, a system must first be effective, and then efficient. This is a macro-milestone that proved to be very adequate to planning and controlling information systems development projects. Steps for achieving effectiveness, and steps for achieving efficiency provide other adequate milestones.

Effectiveness is achieved, as we said above, by thinking and by acting. There are two kinds of thinking: passive (exploratory prototyping) and active (experimental prototyping). In acting, there are two kinds of actors: the analyst/synthetist and the users. Each one of this kind of actors has two kinds of controls: feedback and feedforward. Thus, we have four kinds of actions: user's feedback, user's feedforward, analyst/synthetist's feedback and analyst/synthetist's feedforward (figure 3). Then we



Thinking	
Passive	Active
Exploration	Experimentation

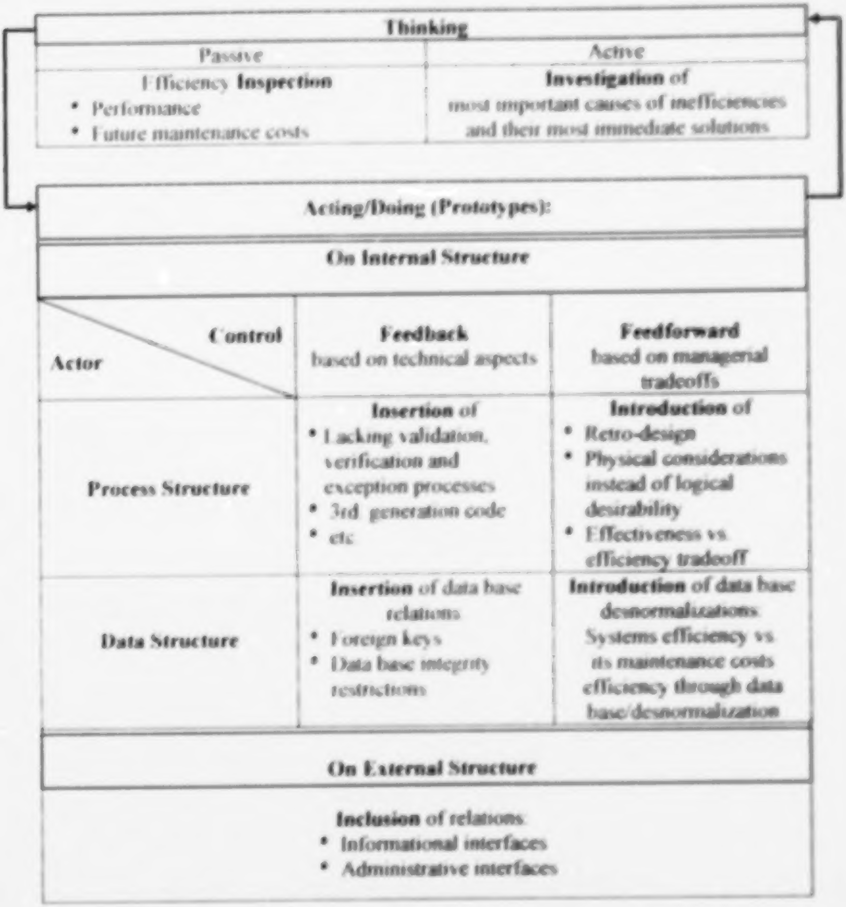
Acting/Doing (Prototypes)		
Control / **Actor**	**Feedback**	**Feedforward**
Users	"Existimation" Opinion/Evaluation of the prototype presented	Expectation Pointing out new expectations for future prototype/system
Analyst/ Synthetist	Extraction of defects and misunderstandings	Expansion To expand the functionality of the prototype

Acting/Using (The System)
System Exploitation

Figure 3

have six activities for achieving effectiveness, which along with the activity of system usage or "exploration", total seven activities oriented to the functionality of the system, i.e. to its external "behavior". We identified seven words beginning with the prefix "ex" to denote the seven concepts related to the activities oriented to the system effectiveness assurance.

Once effectiveness is achieved, then the efficiency of the system is addressed in order to make it operative first, and user friendly later. In a similar way as before we can get seven kinds of activities for achieving system's efficiency as it is shown in figure 4.



Thinking	
Passive	Active
Efficiency **Inspection** • Performance • Future maintenance costs	**Investigation** of most important causes of inefficiencies and their most immediate solutions

Acting/Doing (Prototypes):

On Internal Structure

Control / **Actor**	**Feedback** based on technical aspects	**Feedforward** based on managerial tradeoffs
Process Structure	**Insertion** of • Lacking validation, verification and exception processes • 3rd generation code • etc	**Introduction** of • Retro-design • Physical considerations instead of logical desirability • Effectiveness vs. efficiency tradeoff
Data Structure	**Insertion** of data base relations • Foreign keys • Data base integrity restrictions	**Introduction** of data base desnormalizations: Systems efficiency vs its maintenance costs efficiency through data base/desnormalization

On External Structure

Inclusion of relations:
- Informational interfaces
- Administrative interfaces

Figure 4

There are seven activities related to systems efficiency, which are associated with the system structure. Then, we tried to identify seven words beginning with the prefix "in" in order to denote the seven concepts involved. From here the name of "7 ex-in" of the methodology we have been designing and using.

6. EFFICIENCY AND EFFECTIVENESS OF THE METHODOLOGY "7 EX-IN"

We have been using methodology for five years, through which we developed and implemented about two million lines of code. Consequently, our productivity average along these five years raised to a level 40% higher than the American average, our quality (number of bugs per thousands lines of code the first year of system operation) raised to 55% more than the American average, and our effectiveness (percentage of lines of codes delivered and actually in use) has been 98.5%. More than 30,000 function points has been developed by the methodology described, and all of them are in operation and being used at the present time.

7. REFERENCES

Braybrooke, D., and Lindblom, C.E., (1970), *A Strategy of Decision: Policy Evaluation as Social Process*, The Free Press, New York.

Callaos, N., and Callaos, B., (1991), "A Systemic Definition of Methodology", in S. Holmberg and K. Samuelson (eds.) *Systems Science in the 21st Century*, Proceedings of the 35th Annual Meeting of the International Society for Systems Science, Ostersund, Sweden, June 14-20, Vol. I, pp. 71-78.

Callaos, N. (1992), "A Systemic 'Systems Methodology'", *Proceedings of the 6th International Conference on Systems Research, Informatics and Cybernetics*, Baden-Baden, Germany, August 17-23.

Callaos, N. (1993), "Ethical Considerations for Information and Knowledge Systems Development", in R. Packham (ed.) *Ethical Management of Science as a System*, Proceedings of the 37th Annual Meeting of the International Society for Systems Science, University of Western Sydney, Hawkesbury, Australia, July 5-9, pp. 265-274.

Callaos, N., and Callaos, B., (1994), "A Systemic Methodological Support for Information Systems Analysis and Synthesis", *12th European Meeting on Cybernetics and Systemic Research*, Vienna, Austria, April 5-8.

Callaos, N., (1994), "Conjoined Co-Evolutive Action-Design", *7th International Conference on Systems Research, Informatics and Cybernetics*, Baden-Baden, Germany, August 15-21.

Churchman, C.W. (1971), *The Design of Inquiring Systems*, Basic Books, Inc. Pub., New York.

Mattesich, R. (1978), *Instrumental Reasoning and Systems Methodology: An Epistemology of the Applied and Social Sciences*, D. Reichel Pub. Co., Dordrecht, Holland.

O/A MATRIX AND A TECHNIQUE FOR METHODOLOGY ENGINEERING

Kimmo Kinnunen[1] and Mauri Leppänen[2]

[1]Nokia Telecommunications, P.O. Box 50, SF-44101 Aanekoski, Finland
[2]University of Jyvaskyla, Department of Computer Science and Information Systems,
P.O. Box 35 SF-40351 Jyväskylä, Finland

Abstract

This paper presents a methodology engineering technique for describing, analysing and refining conceptual relationships of the methods and the models within an information system development (ISD) methodology. Methodology engineering (ME) is defined as being composed of the activities of developing, customising and instantiating an ISD methodology to meet organisational needs and to fill the demands of a specific project. A framework of four processing layers and four meta levels is constructed to increase the understanding of the scope and domain of ME. Based on the framework, the Object/Activity (O/A) matrix is suggested as means for describing and analysing the dynamic relationships between the development objects and ISD activities, extracted from the meta data models and the ISD process models, respectively. Further, a ME-technique is provided for the formation, analysis and refinement of an O/A-matrix.

1. INTRODUCTION

Various information systems development (ISD) methodologies have been developed to support the analysis, design and implementation of organisational and technical changes in information processing of an organisation. No end to this expansion of a collection of ISD methodologies is in sight. New methodologies are developed as a consequence of emerging approaches (e.g. object-oriented methodologies), to serve new application areas (e.g. geographical information systems), and also as a response to the experiences gained from applying existing methodologies. These result in changes in collections and contents of models and methods as well as in the logical and temporal structures of ISD processes. Work on a methodology does not end with publishing a book or a manual about it. A methodology has to be customised to achieve the best match with the traditions, maturity, culture and infrastructure of an organisation. Further, it has to be instantiated to meet the specific needs of a particular ISD project. All these activities of development, customisation and instantiation are here called *methodology engineering* (ME). It is a very complicated and multifaceted endeavour. Seen from a conceptual point of view, the purpose of methodology engineering is to analyse, develop and refine an integrated and organised collection of concepts underlying the methods and models of a methodology.

An ISD methodology often contains dozens of models and methods. It is very difficult to between distinguish and describe all their structural, functional and dynamic relationships. Methodology customisation and instantiation make the situation still more complicated; a collection of methods and models evolves and radical changes in their contents may even occur. In spite of these changes, the interoperability of the methods and the models should be guaranteed. In recent years, methodology engineering has been approached through metamodelling. *Metamodelling* is the process of producing a meta model about a model. Most commonly, metamodelling has been applied to individual models (e.g. Godwin et al., 1989). Also some comparisons between models have been made with the aid of meta models (e.g. Brinkkemper et al., 1989). Based on the meta models, customisable CASE tools, called metasystems (Sorenson et al., 1988) or CASE shells (Bubenko, 1988, Smolander et al., 1991) have been constructed. Recently, integrated frameworks to cover, besides the development objects, the ISD-processes and the actors have also been presented (Hahn, Jarke and Rose 1991, Heym and Österle, 1992, Tolvanen et al., 1993).

Although distinguishing and structuring the essential phenomena of the ISD, the existing literature suggest hardly any concrete aid for methodology engineering. It does recognise the structural relationships within and between the meta data models, but totally ignores the dynamic relationships manifested by the ISD process models together with the meta data models.

The purpose of this study is to present a means for describing and analysing the n-ary dynamic relationships between the development objects and the ISD activities extracted from the meta data models and the ISD-process models, respectively. The means for this is the O/A matrix. Further, we suggest a ME technique for forming, analysing and refining an O/A matrix.

The paper is organised as follows. In Section 2, we will construct a comprehensive framework of four processing layers and four meta levels to increase the understanding of the scope and the domain of ME. Based on this framework we define the concept of interoperability between the models and methods. In Section 3, we will describe a ME technique based on the O/A matrix and illustrate its use by examples taken from the analysis of a methodology. In Section 4 some conclusions are presented.

2. METHODOLOGY ENGINEERING

An ISD methodology is seen, from a conceptual point of view, as a complicated aggregate of conceptual constructs that are composed of concepts referring to philosophies, approaches, principles, models, methods and techniques. A methodology is often a result of extensive and laborious engineering work. Methodology engineering comprises a large variety of activities divided into three categories: development, customisation and instantiation. By *development* we mean the R&D activities, carried out by research institutes, universities, and consulting companies, to produce and publish detailed descriptions of a methodology. This kind of methodology we call a general methodology (e.g. SSADM (Skidmore et al., 1992), SSA (Gane and Sarson, 1979), ISAC (Lundeberg et al., 1980), SDM (Turner et al., 1988)). By *customisation* we mean those activities that are carried out to make a methodology suitable for an organisation. On the basis of the contingencies concerning the organisational culture, maturity, methodology experiences, facilities, etc., the most suitable models and methods are selected, tailored, and described as a customised methodology in a manual that is internal to an organisation. By *instantiation* we mean the activities that select those models and methods that are applicable to a particular ISD project. Further, they structure and schedule the development processes, decide on milestones and deliverables, and assign persons to positions with certain rights and responsibilities. Besides written form, a methodology may appear as a computer-aided development environment resulting from the implementation of a methodology. Exploitation of such environment makes the customisation and instantiation of a methodology much easier.

2.1. Framework

All the decisions made during methodology engineering aim to achieve the "best" collection of executable principles, methods, models and techniques for an ISD project. The ultimate quality assessment of a methodology cannot be made until adequate experiences from the applications of a methodology in different situations have been gained. Several decisions and assessments can, however, be made during an engineering process. Metamodelling is valuable means to this work. Metamodelling means a process of producing a meta model. In the following, we will apply the universal framework for information activities by Auramäki et al. (1988) to build a conceptual framework to define the concepts of meta model and metamodelling. The framework consists of processing layers and meta levels (Figure 1).

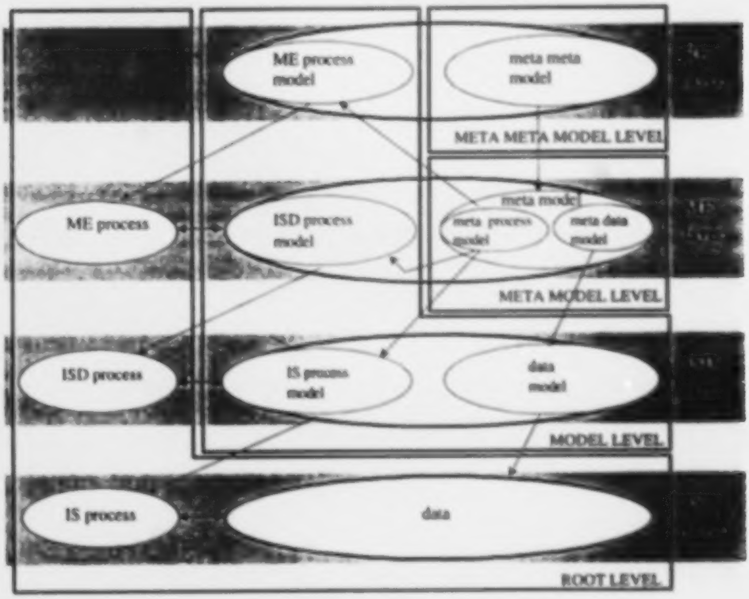

ME process model
meta meta model
META META MODEL LEVEL
meta model
ME process
ISD process model
meta process model
meta data model
META MODEL LEVEL
ISD process
IS process model
data model
MODEL LEVEL
IS process
data
ROOT LEVEL

Figure1. Framework

The *processing layers* are the information system layer (IS layer), the information system development layer (ISD layer), the methodology engineering layer (ME layer) and the ME research layer (RE layer). On each processing layer, we distinguish two kinds of phenomena, processes and objects, with the meaning that the *processes* create, change, transmit, verify, and utilise the *objects*. The objects on a upper processing layer describe and prescribe the objects on the (next) lower processing layer. On the IS layer, there are *IS-processes* and *(application) data*. They are described/prescribed by an *IS process model* and a *data model*, respectively. An IS-process model and a data model are the *ISD objects* on the ISD layer and processed by *ISD processes*. The ISD objects are described/prescribed by a *meta process model* and a *meta data model*, and the ISD processes are described/prescribed by an *ISD process model*. The ISD process model and the meta models are the objects processed by ME processes on the ME layer. ME processes are executed according to a *ME process model*. The meta models are described/prescribed by a *meta meta model*. The meta meta model and the ME process model are research objects on the RE layer.

The *meta levels* are the root level, the model level, the meta model level, and the meta meta model level. The relations between them are instance_of -relations, i.e. a concept on an upper meta level is a type concept for (instance) concepts on the (next) lower meta level. Thus the data on the root level, for example, is an instance of a data model (e.g. an ER schema) on the model level, and a process model (e.g. data flow diagram) is an instance of a meta process model on the meta model level. The same or different meta process model can be used to produce an ISD process model and a ME process model. It is noteworthy that for this part the meta levels differ most from the layers. One or more meta meta models on the highest level can be used as a basis for producing a meta process model and a meta data model.

A *method* is a set of principles and procedures that prescribe how to produce or implement a model. A *technique* is a stepwise description of processes needed to carry out a development task. The description of a method comprises a meta model and an ISD process model. A methodology is composed of a large variety of meta models, ISD process models and meta meta models (cf. Figure 1). Models, meta models and meta meta models are expressed in some languages. A language itself can be described by a (meta) model.

2.2. Interoperability

ISD and ME involve a great number of models and methods. From the viewpoint of practical work, it is of great importance that all these models and methods are interoperable. This is the only way we can ensure that the necessary processes are carried out properly and in the right order. To illustrate the nature of the interoperability, we will consider more closely in the following the meta data models, the ISD process models and their relationships (Figure 2).

The meta data models reflect a structural point of view upon an ISD project. They reveal the objects and their relationships. The process models adopt a functional point of view in distinguishing ISD processes, ISD deliverables and their relationships. On the basis of this simplified division, we can now define the concepts of structural and functional interoperability.

The *structural interoperability* of an ISD methodology measures the consistency of the objects of its models and methods and their structural relationships. The most common relationships can be manifested by generic specifications. For instance, a bridge from a data flow diagram to an ER schema can be built through the structural relationship "Is concerned" between "Data storage" and "Entity type/Relationship type". Besides generic relationships, there are more specific structural relationships that are based on mutualism, inheritance, or polymorphism. A mutual relationship exists between two methods if their object sets intersect (e.g. the object "Information set" is mutual for the I-graph and the C-graph in the ISAC methodology, Lundeberg et al., 1980). An inheritance relationship holds between two objects one of which is a sub-object of the other object. A polymorphic relationship exists between objects that are modelled by different concepts (e.g. as an entity and as a property) in two meta data models although they refer to the same (kind of) phenomenon in reality.

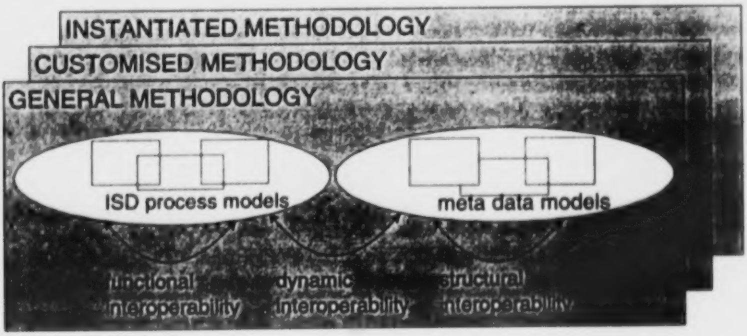


Figure 2. Interoperabilities.

The *functional interoperability* of an ISD methodology measures the consistency of the functional relationships between the concepts of the process models. The functional relationships become established in two ways: (a) two process models share the same process or a part of it, or (b) two process models contain a deliverable that is described as an output in one process model and as an input in the other process model.

The meta data models basically give a static view on ISD. They do not explicitly regulate how and in which order objects should be distinguished, described and analysed. However, they do reflect, implicitly, the intrinsic logical relationships between the objects, and that should have an affect on the dynamics of an ISD methodology. Let us illustrate this by two examples. In conceptual modelling, it is reasonable first to "Recognise entity types and relationship types" and after that to "Define attributes" because, according to the meta data model of the ER-model, the object "Attribute" is a weak object that should always be connected to the object "Entity type" or "Relationship type". Second, the object "Storage structure" in a method of physical database design is a weak object for "Relation" that is an object of a logical database design method. Therefore, it would be inconsistent to state that a storage structure for a relation is to be decided before a relation scheme is defined.

A dynamic view, formed merely by the integration of the separate process models about the methods, does not involve the logical relationships mentioned above. A dynamic view in a real sense is not reached until the logical relationships are taken into account. To enable the analysis of logical relationships, we define the concept of dynamic interoperability as follows: the *dynamic interoperability* of an ISD methodology measures how consistent the functional relationships between the concepts of the process models are with the logical relationships between the concepts of the meta data models.

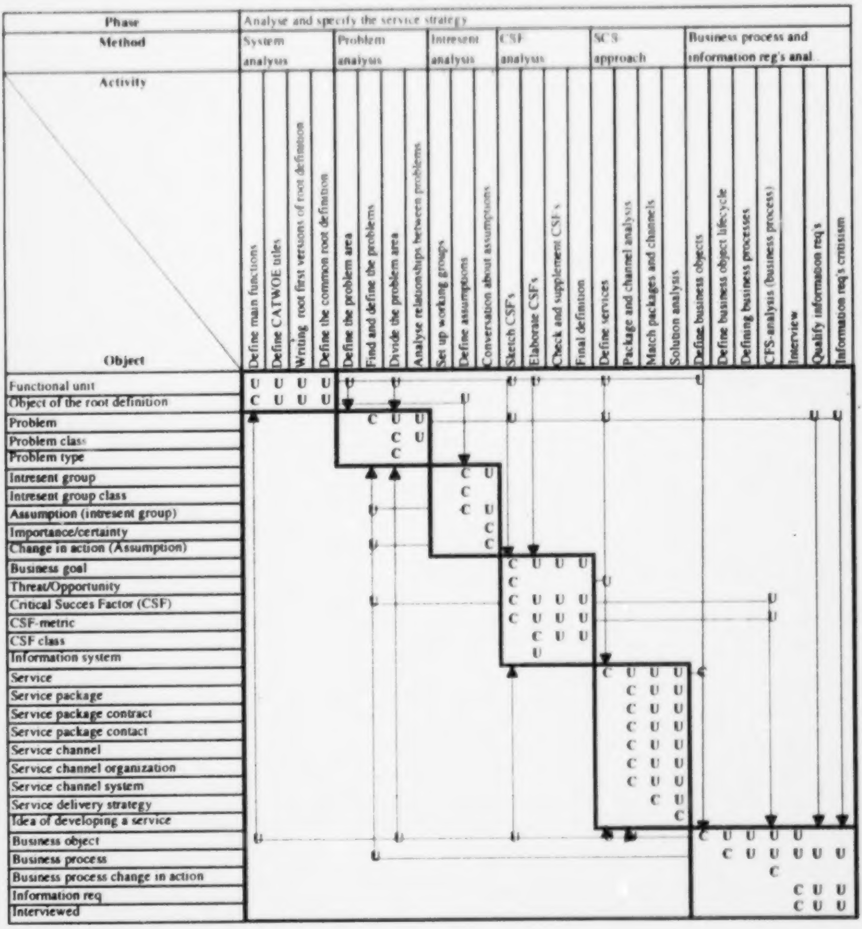
Figure 2 presents three frames for the descriptions of a methodology, one for a general methodology, one for a customised methodology and one for an instantiated methodology. Our discussions above apply to all the frames. In each phase of the methodology engineering it is equally important to ensure the structural, functional and dynamic interoperability of a methodology.

The relationships between the meta data models and the process models can be specified and analysed on several levels of abstraction. Some approaches (e.g. Brinkkemper et al., 1989) attempt to specify binary relations between two methods by building formal "bridges" between the meta data models and the process models of the methods. This requires a lot of effort. Besides, the relations tend to remain fixed thus complicating the customisation and instantiation of the methods. We take a different approach. Our purpose is to develop means to describe in a flexible and easy way the n-ary relationships between the methods and the models on a higher level of abstraction. This description should provide a platform for the analysis of the dynamic interoperability of the method and models of a general, customised and instantiated methodology. For this purpose we suggest the O/A matrix. In the following section, we will describe the matrix and a technique we have developed to form, analyse and refine it.

3. ME TECHNIQUE

The Object/Activity (O/A) matrix has been developed from the process/data class matrix introduced in the BSP methodology (IBM, 1984). It is used here to express the relationships between the objects and the activities extracted from the meta data models and the ISD process models. Object are things in the universe of discource that are perceived, from the viewpoint of some method, as relevant and existing on their own. From the viewpoint of Problem Analysis, for instance, Objects are "Problem", "Business Process" and "Functional Unit". A method is decomposed into activities and further into steps. One or more methods constitute a phase of a methodology. The matrix expresses two kinds of relationships between objects and activities: creation relationships and usage relationships. A *creation*-relationship (C) means that an object is fundamental for the activity concerned. An object is *fundamental* if an activity of a method provides specific means for collecting, presenting and analysing knowledge about it. A *usage* relationship (U) means that some knowledge about the object, provided by another activity, is used by that activity. The letters C and U are placed into the cells of the matrix. If a cell has no letter, the activity has no relationships with the object. In Figure 3, an example of an O/A matrix is presented.

A methodology engineering technique (ME technique) has been developed to form, analyse and refine an O/A matrix. The basic idea in the refinement of an O/A matrix is to remove those usage relationships that conflict with the dynamic interoperability of the methods. The technique is independent from the language used to present meta data models and ISD process models, provided that it recognise objects, relationships between the objects and activities. In this paper, we use the GOPRR meta language (Tolvanen et al., 1993). The concepts and constructs of the language are defined in Appendix 1. To illustrate the language and its use the meta data model and the ISD process model of one method (Intresent analysis by Mason and Mitroff, 1984) are also included in the appendix.



Phase	Analyse and specify the service strategy																									
Method	System analysis				Problem analysis				Intresent analysis			CSF analysis				SCS approach				Business process and information req's anal.						
Activity / Object	Define main functions	Define CATWOE titles	Writing root first versions of root definition	Define the common root definition	Define the problem area	Find and define the problems	Divide the problem area	Analyse relationships between problems	Set up working groups	Define assumptions	Conversation about assumptions	Sketch CSFs	Elaborate CSFs	Check and supplement CSFs	Final definition	Define services	Package and channel analysis	Match packages and channels	Solution analysis	Define business objects	Define business object lifecycle	Defining business processes	CFS-analysis (business process)	Interview	Qualify information req's	Information req's critisism
Functional unit	U	U	U	U	U		U					U	U			U				U						
Object of the root definition	C	U	U	U						U																
Problem						C	U	U				U				U									U	U
Problem class							C	U																		
Problem type							C																			
Intresent group										C	U															
Intresent group class										C																
Assumption (intresent group)						U				C	U															
Importance/certainty											C															
Change in action (Assumption)						U					C															
Business goal												C	U	U	U											
Threat/Opportunity												C				U										
Critical Succes Factor (CSF)						U						C	U	U	U								U			
CSF-metric												C	U	U	U								U			
CSF class													C	U	U											
Information system													U													
Service																C	U	U	U	C						
Service package																	C	U	U							
Service package contract																	C	U	U							
Service package contact																	C	U	U							
Service channel																	C	U	U							
Service channel organization																	C	U	U							
Service channel system																	C	U	U							
Service delivery strategy																		C	U							
Idea of developing a service																			C							
Business object	U						U					U								C	U	U	U	U		
Business process						U															C	U	U	U	U	U
Business process change in action																							C			
Information req																								C	U	U
Interviewed																								C	U	U

Figure3. Initial O/A matrix of the first phase of the SPITS methodology.

We have deployed the ME technique to analyse the SPITS methodology (Strategic Planning of Information Technology Services) (Leppänen et al., 1991). It has been developed to support analysis and specification of the service strategy, the planning of the information technology strategy, and its implementation. The methodology is a composition of well-known methods, mainly originating from the field of strategic information systems planning. It is organised into three phases. In this paper, we will focus on the analysis of the first phase: analyse and specify the service strategy. The phase is composed of six methods: System analysis (Checkland, 1981), Problem analysis (Goldkul and Röstlinger, 1988), Intresent analysis (Mason and Mitroff, 1981), Critical success factors (CSF) analysis (Rockart, 1979), Service Channel Strategies (SCS) Approach (Mäkelin and Vepsäläinen, 1989), and Business process and information requirements analysis (Blokdijk and Blokdijk, 1987). There are two reasons for the application of the ME technique to the analysis of the SPITS methodology. First, the methodology is still in an early phase of its development life cycle and thus the analysis is expected to result in actual improvements in the dynamics of the methodology. Second, the SPITS methodology, as a composition of a large variety of methods, is expected to contain just those problems that the ME technique has been developed to solve.

The ME technique is structured into four steps by which an O/A matrix is to be formed, analysed and refined. The steps can be applied in a successive, parallel or iterative fashion. In the descriptions of the steps below we will refer to the O/A matrices in Figures 3 and 4:

Step 1. Place the phases, methods and activities into the columns of the O/A matrix in a logical order. Some methodologies are strictly structured and provide the unambiguous order of phases, methods and activities (cf. Lundeberg et al., 1981). Most of the modern methodologies (e.g. Olle et al., 1982, Olle et al., 1986), however, do not take a stand on how exactly activities should be ordered. This is because the methodologies are to be applied in different ways in different situations. However, there are some intrinsic logical relationships among the methods and the activities that should be recognised and indicated by the column positions in the matrix. These relationships form the *logical* order into which the methods and the activities extracted from the ISD process models are to be located.

Step 2: Place the objects into the O/A matrix and indicate their relationships with the activities. Examine each method and activity, following the specified logical order, in order to extract essential objects from each meta data model. Insert them into the rows of the O/A matrix if they are not already there. Indicate the way an activity deals with an object by putting the letter C or U into the corresponding cell. For example "Intresent group" is a fundamental object for the activity "Define assumptions" because different intresent groups are recognised and described by that activity. Therefore, the letter C is located into the corresponding cell. Descriptions and definitions about the object "Intresent Group" are utilised by the activity "Conversation about assumptions". Besides "Intresent group", the activity "Define assumptions" 'creates' the objects "Intresent group class" and "Assumptions". Finding relationships and deciding on their nature is clearly the most laborious task in this work.

Step 3: Sketch the scopes of, and the interrelations between, the methods. Indicate the scope of each method by embracing the relationships between the objects and the activities by boxes. The vertical sides of a box correspond to the borderlines of the method. The horizontal sides are determined by the positions of those objects that are attached through the creation relationships to some activity of the method. The relationships that remain outside the boxes manifest the interrelations between the methods. Arrows are used to show the directions of the relationships. An arrow in the upper triangle expresses that an object is used by an activity that is to be performed after the activity that creates the object. An arrow in the lower triangle expresses that an activity using the object is to be performed before the activity, which creates that object. After drawing the boxes and the arrows the *initial matrix* is ready (Figure 3).

Step 4: Analyse and refine the matrix. Based on the initial matrix several conclusions can be made about the coverage and interoperability of the methods. These can result in plenty of refinements until the final matrix is reached:

(a) If an object is attached through creation relationships to several activities and to several methods, the division of ISD processes into methods and activities should be reconsidered.

(b) If an object has no creation relationship to any activity, it is possible that (i) a collection of methods within a methodology is inadequate, or (ii) the knowledge about the object is received as an input from the activities that are covered by some other methodology applied in parallel in an organisation. Examples of the latter case are the objects "Information system" and "Functional unit".

(c) If some activity has usage relationships (to activities of some other method) that are expressed by arrows in a lower triangle, it manifests conflicts with the requirement of the dynamic interoperability of the methods. For instance, in the initial matrix of Figure 3 "Assumption" created by Intresent analysis is used by Problem analysis. These kinds of relationships should be removed. There are two means to refine the matrix: (i) change the place of the method in the logical order, or (ii) split the method into two or more parts and relocate them. The former principle should be preferred. Sometimes conflicting relationships cannot be avoided. That implies a strong iteration or parallelism between the methods. In Problem analysis, for instance, potential problems are searched for by analysing the critical success factors. The parallelism of the methods becomes evident from two relationships between the objects "Problem" and "CSF". They form a cycle.

We applied the technique to the analysis of the SPITS methodology through three iterations. In the following, some examples about the findings and refinements of the initial matrix are mentioned. The first finding from the analysis was that Problem analysis has conflicting relationships with Intresent analysis involving the objects "Assumption" and "Change in action". These were avoided by changing the positions of the methods. Second, it was found out that the activities of "Business process and information requirements analysis" have some conflicting relationships, involving the objects "Business object" and "Business process", with some activities of the methods placed as to be performed before "Business process and information req's analysis". The same activities have, however, several relationships that are in accordance with dynamic interoperability. Because no other objects of the method were involved by the conflicting relationships, the method was split into two parts: "Business process analysis" and "Information req's analysis". "Business process analysis" was moved to be the second method. Because "Service" is a fundamental object for "Business process analysis", its position was changed among the rows. As a result of refinements, all conflicting relationships, except one, were removed. The relationship between the objects "Problem" and "CFS" remained as an indication of parallelism of the concerned methods. The final matrix is presented in Figure 4.

There are still three remarks to be made about the application of the ME technique. First, the analysis and refinement of an O/A matrix is possible although the order between the methods and the activities, as a result of Step 1, would be almost random. This means that the technique can also be applied to establish a rough life cycle of a methodology in an initial stage of methodology development. Second, two actions of refinements were suggested: moving and splitting. Besides these, it is naturally possible to compose methods or their components into new methods in order to avoid conflicting relationships. This is not discussed here since we want mainly to concentrate on methodology engineering, not method engineering. As an extreme case of this, we could place into a matrix the activities extracted from the methods quite independently of the original borderlines of the methods and derive their scopes from the expressed relationships between the objects and the activities.

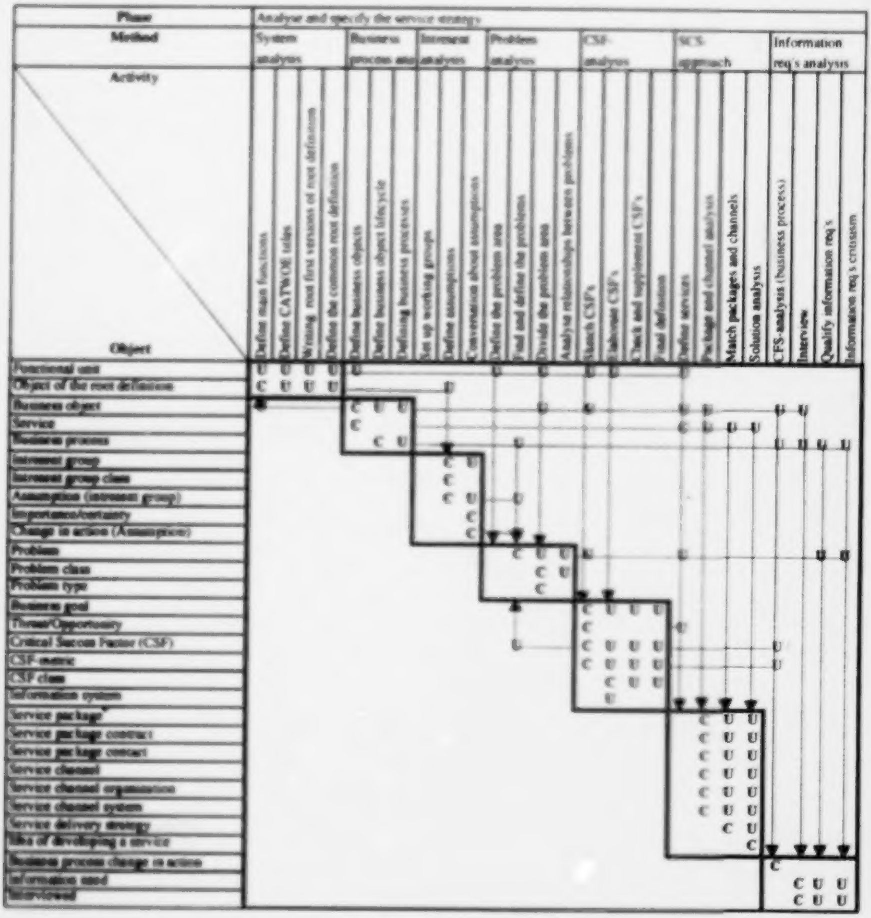

Phase
Method
Activity
Object
Analyse and specify the service strategy
System analysis
Business process analysis
Interest analysis
Problem analysis
CSF-analysis
SCS-approach
Information req's analysis
Define main functions
Define CATWOE tasks
Writing root first versions of root definition
Define the common root definition
Define business objects
Define business object lifecycle
Defining business processes
Set up working groups
Define assumptions
Conversation about assumptions
Define the problem area
Find and define the problems
Divide the problem area
Analyse relationships between problems
Match CSF's
Elaborate CSF's
Check and supplement CSF's
Final definition
Define services
Package and channel analysis
Match packages and channels
Solution analysis
CFS-analysis (business process)
Interview
Qualify information req's
Information req's criticism
Functional unit
Object of the root definition
Business object
Service
Business process
Interest group
Interest group class
Assumption (interest group)
Importance/certainty
Change in action (Assumption)
Problem
Problem class
Problem type
Business goal
Threat/Opportunity
Critical Success Factor (CSF)
CSF-metric
CSF class
Information system
Service package
Service package contract
Service package concept
Service channel
Service channel organisation
Service channel system
Service delivery strategy
Idea of developing a service
Business process change in action
Information need
Interviewed
U
C
U

Figure 4. Final O/A matrix of the first phase of the SPITS methodology.

Third, we apply the framework in Figure 1 to summarise our description of the O/A matrix and the ME technique. Our purpose has been to develop and describe a ME technique to be included into the ME process model. The basic idea behind this technique

BEST COPY AVAILABLE

is to incorporate, on a rather high level of abstraction, the meta data models and the ISD process models, to analyse and improve their interoperability. We have used the GOPRR model as our meta meta model. Actually, we have applied the same meta meta model to present the meta data model of the O/A-matrix and the ME process model of the ME technique. Unfortunately, it is not possible to present them within the limits of this paper.

4. CONCLUSIONS

In this paper we have discussed methodology engineering (ME) and suggested a technique to support it. Methodology engineering was defined as being composed of the activities of developing, customising and instantiating an ISD methodology to meet organisational needs and to fill the demands of a specific project. A framework of four processing layers and four meta levels was constructed to help in perceiving and structuring the scope and domain of ME. Based on the framework, the meta data models and the ISD-process models were discovered as the most essential components for ME work. Dynamic interoperability was defined as the consistency of the relationships between the meta data models and the ISD process models. The O/A matrix was suggested as means of describing the n-ary relationships between the meta data models and the ISD process models. Further, a ME technique for the formation, analysis and refinement of an O/A matrix was presented and illustrated by examples taken from the analysis of the SPITS methodology.

The ME technique, based on the O/A matrix, is mainly intended for the analysis of the dynamic interoperability of the models and the methods within a methodology. The analysis may cause changes in the borderlines between, and in the execution order of, the methods. The technique can be implemented as a tool for aiding the management of a CASE shell method base. The computer-aided technique can be included in the environments (e.g. CAME, Kumar and Welke, 1992) which provide domain knowledge about an IS and ISD work to support intelligent reasoning about the applicability of the methods and the models to a specific project.

REFERENCES

Auramäki, E., Leppänen, M. and Savolainen, V., (1988), Universal framework for information activities, *Data Base*, ACM, Vol. 19, No. 1, pp. 11-20.

Blokdijk, A. and Blokdijk, P., (1987), "Planning and Design of Information Systems", Academic Press, London.

Brinkkemper, S., Geurts, M., van de Kamp, I. and Acohen, J., (1989), On a formal approach to the methodology of information planning, in: "Proceedings of the First Conf. on Information Systems" (ed. Maes, R.), Amersfoort.

Brinkkemper, S., de Lange, M., Looman, R. and van der Steen, F.H., (1989), On the derivation of method companionship by metamodelling, in: "CASE'89 Third International Workshop on Computer-Aided Software Engineering", London, IEEE Computer Society Press, 266-286. Bubenko, J., (1988), "Selecting a Strategy for Computer-Aided Software Engineering (CASE)", SYSLAB-Report nr. 59, University of Stockholm, Sweden,Lund.

Checkland, P., (1981), "Systems Thinking, Systems Practice", John Wiley & Sons, Chichester.

Gane, C. and Sarson, T., (1979), "Structured System Analysis: tools and techniques", Prentice Hall, Englewood Cliffs.

Godkuhl, G. and Rlstlinger, A., (1988), "Förändringsanalys - Arbetsmetodik och Förhållnigssätt för Goda Förändringsbeslut", Studentlitteratur, Sweden.

Godwin, A.N., Gleeson, J.W. and Gwillian, D., (1989), An assessment of the IDEF notations as descriptive tools, *Information Systems*, Vol. 14, No. 1, pp. 13-28.

Hahn, U., Jarke, M. and Rose, T., (1991), Teamwork support in knowledge-based information system environment, *IEEE Trans. on Software Engineering*, Vol. 17, No. 5, pp. 476-481.

Heym, M. and Österle, H., (1992), A semantic data model for methodology engineering, in: "Proceedings of the Fifth CASE '92 Workshop" (eds. Forte, G., Madhavji, N.), Montreal, IEEE Computer Society Press, Los Alamitos.

IBM, (1984), "Business Systems Planning, Information Systems Planning Guide", GE20-0527-4, IBM Corporation, Atlanta.

Kumar, K. and Welke , R.J., (1992), Methodology engineering: a proposal for situation specific methodology construction, In "Challengies and Strategies for Research in Systems Development" (eds. Cotterman, W.W. and Senn, J.A.), John Wiley & Sons Ltd, New York, 257-269.

Leppanen, M., Lyytinen, K. and Halttunen, V., (1991), "Strategic Planning of Information Technology Services (SPITS)" (in Finnish), University of Jyväskylä, Dept. of Computer Science and Information Systems, Computer Science Reports TU-10, Jyväskylä.

Lundeberg, M., Goldkuhl, G. and Nilson, A., (1980), "Information Systems Development -a Systematic Approach", Prentice Hall, Englewood Cliffs.

Mason, R. and Mitroff, I., (1979), "Challenging Strategic Planning Assumptions, John Wiley & Sons, New York.

Mäkelin, M. and Vepsäläinen, A., (1989), "Service Strategies - Developing Service Channels with Information Technology" (in Finnish), HM&V Research Oy, Helsinki, Finland.

Olle, T., Sol., H. and Tully, C. (eds.), (1983), "Information Systems Design Methodologies: a Feature Analsysis", North-Holland, Amsterdam.

Olle, T., Sol., H. and Verrij-Stuart, A. (eds), (1982), "Information Systems Design Methodologies: a Comparative Review", North-Holland, Amsterdam.

Rockart, J., (1979), Chief executive define their own information needs, *Harward Business Review*, March-April 1979.

Skidmore, S., Farmer, R. and Mills, G., (1992), "SSADM - Version 4: Models & Methods", NCC Blackwell, Oxford.

Smolander, K., Lyytinen, K., Tahvanainen, V.P. and Martiin, P., (1991), MetaEdit - a flexible graphical environment for methodlogy modelling, in "Proc. of the Third International Conference CAiSE91", LNCS 498, pp. 168-193.

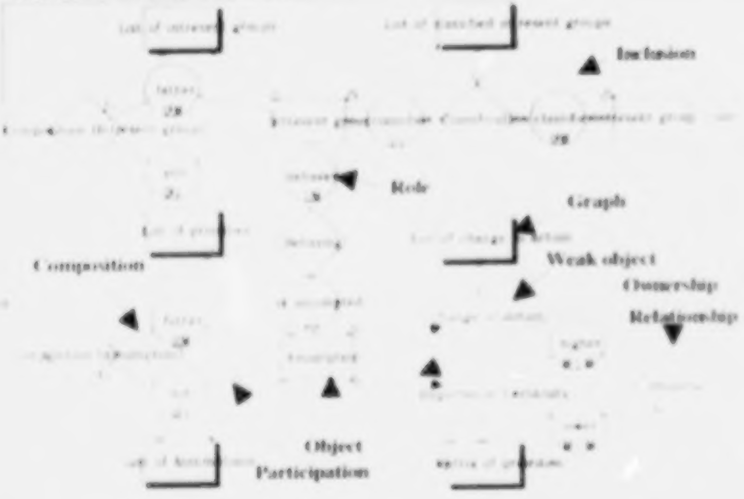
Sorenson, P., Tremplay, J.-P. and McAllister, A., (1988), The Metaview System for many spefication environments, *IEEE Software*, March, pp. 30-38.

Tolvanen, J., Marttiin, P. and Smolander, K., (1993), An integrated model for information systems modeling, in: "Proceedings of the 26th HICSS", Vol. 3, (eds. Nunamaker, J.F. jr and Sprague, H.M.), Hawaii, January 1993, pp. 470-479.

Turner, W.S., Langerhorst, R.P., Hice, G.F., Eilers, H.B. and Uijttenbroek, A.A., (1988), "SDM - System Development Methodology", North-Holland, Amsterdam.

Appendix 1. An example of metamodels (GOPRR)

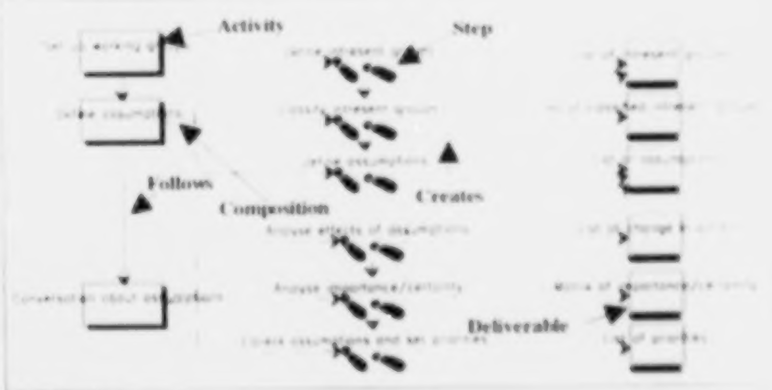
Meta data model of intresent analysis


Inclusion
Role
Graph
Composition
Weak object
Ownership
Relationship
Object
Participation

Object types
- Object - a thing with independent existence
- Weak object - an object that is dependent on another object
- Relationship - connects object in a meaninful way through roles
- Role - specifies how an object participates in a relationship
- Property - describe/qualify characteristic associated with the other types
- Graph- collection of all other types

Relationship types
- Participation, Aggregation, Composition, Inheritance, Inclusion, Ownership, Explosion

Process model of intresent analysis


Activity
Step
Follows
Composition
Creates
Deliverable

Object types
- Decision - steers a planning process (Milestone,Co-ordination)
- Task - performs a planning process (Activity, Step, Checking, Taansformation, Review)

Relationship types
- Follows, Composition, Uses and Creates

BEST COPY AVAILABLE

METAMODELLING: CONCEPT, BENEFITS AND PITFALLS

Mauri Leppänen

University of Jyväskylä, Department of Computer Science and Information Systems
P.O. Box 35 SF-40351 Jyväskylä, Finland

Abstract

Metamodelling refers to the process of producing a meta model to describe or prescribe the conceptual structure or the language of a model. In this paper a comprehensive conceptual framework is first built to promote the undertanding of the concepts of a model and modelling. This framework is derived from the concept of a object by applying a semiotic approach. Several classifications of models and modelling activities are presented. Based on these underpinnings concepts of a meta model and a metamodelling are defined. A meta structure model and a meta process model are distinguished and the activities to produce them are discussed. Finally, benefits and pitfalls of metamodelling are reviewed.

1. INTRODUCTION

The goal of *information system development* (ISD) is to analyse, design, and implement organisational and technical changes in the information processing of an organisation, in order to improve an organisation's efficiency and effectiveness and/or to reshape its competitive position in the market. During the ISD, numerous models about activities, people, information sets, facilities and their multifaceted relationships are produced. These are called information system models (IS models). ISD follows a number of prescriptions that together constitute a whole called an ISD-methodology. An ISD methodology is a model as well. It prescribes on what, when, why and how people involved in the ISD should work to make the IS successful. And on the level still on higher, we can find models (meta models) which describe the concepts and the conceptual constructs used to establish an ISD methodology.

To analyse, compare, select, elaborate and manage a huge variety of models, meta models need be produced and applied. Meta models describe the concepts in models and/or the language(s) used to represent the models. With the aid of meta models it is possible to concentrate on the essentials of models and their relations (Saeki et al., 1993, van Slooten and Brinkkemper, 1993). Metamodelling refers to the process of producing meta models.

Although metamodelling is not a new idea, there are only few discussions which provide a firm conceptual basis for the understanding of the concepts of a model and a meta model. The conceptual framework by Frisco Task Group (Lindgren, 1990) is one of the most exhaustive studies on the fundamental concepts of information systems, but it does not, however, cover the information system development, nor the ISD methodologies. Bergheim et al. (1989) include the notion of a meta concept in their taxonomy of concepts for the science of information systems. But their discussion about four meta levels of models and methods does not concern a meta model neither metamodelling. Brinkkemper (1990), one of the pioneers on metamodelling, uses the term 'meta' in an inconsistent manner when discussing the formalization of modelling. Besides presenting misconceptions, the ISD literature falls short of providing a systematic outlook of the potentials and the limitations of metamodelling.

In this paper, a comprehensive conceptual framework is first built to promote the understanding of the concepts of a model and a modelling (Section 2). The framework is derived from the concept of an object by applying a semiotic approach. The approach suggests that models are defined as conceptual and linguistic objects referring to the phenomena in the universe of discourse (UoD) (Section 3). Grounding on this basis, the concepts of a meta model and a metamodelling are defined and specialized into several subconcepts (Section 4). Finally, we will show how one can benefit from the use of meta models and caution about possible misuses of metamodelling (Section 5).

2. CONCEPTUAL FRAMEWORK

Reality is anything that exits, has existed or will possibly exist. From the philosophical viewpoint, reality can be divided into two parts, subjective and objective. The former is one which is observed, perceived or otherwise mentally produced by a human being, and it is thus dependent on the human mind. The latter is independent of any human being. It is the source of sense data which we obtain, and thus it is external to us.

An *object* is any phenomenon in the objective or subjective reality. It can be a concrete or an abstract object. The human mind produces different subjective objects from the same objective reality, depending on the point of view adopted. Using a *point of view* some of the aspects of an object are selected because they are more relevant than others. When a statement is made from that point of view then the reasons for the statement are just selected aspects (Hautamäki, 1986). To derive and relate views, a framework is generally deployed. A *framework* is an organized set of concepts, conceptions and circumstances within which the objects of interest appear according to the human mind (Webster, 1980). It guides human being to select the point of views that are the most appropriate for the case or the problem[1]. Examples of the rigid frameworks are the semiotic framework and the intension/extension frameworks applied in the following.

The semiotic framework is based on the theory of signs - semiotics (Morris, 1964). In semiotics, three realms are distinguished: a realm of concepts, a realm of referents, and a realm of signs. The first realm belongs to the subjective reality while the second and third realms are parts of the objective reality. By applying the meaning triangle of Ogden and Richards (1923), the semiotic framework can be illustrated like in Figure 1.

Using the semiotic framework, we can distinguish between three kinds of objects: concept objects, referent objects and sign objects. *Concepts* are mental constructs, words

---

[1] Note that a framework and a point of view are objects, too.

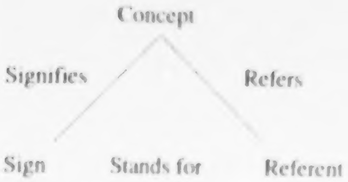


Figure 1. The meaning triangle as a semiotic framework

of mind, by means of which the mind apprehends or comes to know things. They are basic epistemological components of human knowledge. *Referents* are phenomena in reality to which concepts refer. They can be physical things, processes, events, or like. *Signs* or symbols are anything which can stand for something else. They are representations of concepts expressed in a symbolic or iconic language. In Figure 1 the associations between concepts, referents and signs are shown by the sides of the triangle. A sign signifies or designates a concept. A concept signifies or refers to a referent. A sign stands for a referent.

In the following, the term 'object' is used to denote a referent whenever there is no danger of confusion. Otherwise, more precise terms like 'sign object', 'concept object' and 'referent object' are used. To distinguish between the names of the objects, quotation marks are used in the following way. Sign objects are enclosed in simple quotes (e.g. 'John'). The names of referent objects are expressed in douple quotes (e.g. "John"). The names of concept objects are expressed without any quotes (e.g. John).

The intension/extension framework is based on the well-known theories in philosophy and linguistics (Lyons, 1977). In the following, the framework is applied as it is conceived in philosophy. The intension or comprehension of a concept consists of all its characteristics. Characteristics are concepts which are used to characterize the concept. For instance, the concept Animal is a characteristic of the concept Cat. An intension makes up the idea, and none of its characteristics can be removed without destroying the idea (Arnauld, 1964). A core intension of a concept consists of all those specific characteristics (earmarks) that are essential to handle the concept (Bunge, 1967). Characteristics determine the applicability of the concept.

An *extension* of a concept is the set of all (referent) objects to which the intension of the concept applies. The set of referent objects is called an *object set*. If two concepts have the same intension, then they always have the same extension. Two concepts may have the same extension but have different intensions.

For some concept, one corner of the meaning triangle may be absent. The concepts with no referent objects are called *abstract concepts*. The other concepts are called *concrete concepts*. The concepts which can only refer to one object are called *individual concepts* or particulars. The concepts referring to several objects are *generic concepts* or universals.

In the fields of data bases (e.g. Chen, 1976), information systems (cf. Olle et al., 1982) and knowledge representation (cf. Meersman et al., 1990), the concepts interrelated by instance-of -relationships are commonly used. These concepts are here called type concepts and instance concepts[2]. Person, for instance, is a type concept and John is its instance concept. The relation between a type concept and an instance concept can be specified extensionally and intensionally. From the extensional viewpoint, the object refered by an instance concept is a member of the object set of the type concept. From the intensional viewpoint, a type concept states through its intension what kinds of properties are required to regard an object as its instance concept. The principle by which a type concept is generated from the instance concepts is called *classification*. The inverse principle is called *instantiation*.

A type concept may be regarded, from the other point of view, as an instance concept of some other type concept which is called a *meta type concept*. A meta type concept states the properties that a type concept has to meet in order to be seen as an instance concept of that meta type concept. The concept of an Entity type is commonly defined as the specification of all those characteristics that are common to the same kind of entities (objects) with independent existence. This implies that Person is an Entity type. Correspondingly, we can find a concept for which a meta type concept is an instance concept. This concept could be called a meta meta type concept. The concept of a Concept , for instance, is a meta meta type concept for Entity type.

Hence, we can conclude by saying that the principle of classification generates a conceptual hierarchy within which the concepts are interrelated with one another by the *instance_of* -relations.

3. MODEL AND MODELLING

A *model* is an object that is used to help or enable the analysis, design, and/or implementation of some other object[3]. We distinguish between two main types of models: concrete models and abstract models (Figure 2). The *concrete models* are composed of concrete objects. They are linguistic models or non-linguistic models. The ***linguistic models*** comprise signs or symbols represented according to the syntactic and semantic rules of some language(s) on some data carrier (e.g. like on a paper, a screen, or a black-board)[4]. The *non-linguistic models* consist of physical elements which, as an organized composition, resemble some other object(s) (e.g. small copies of machines or ships). The *abstract models* are conceptual structures in human minds.

Figure 2 presents the models and the processes of producing the models (cf. Brinkkemper, 1990). The non-linguistic models are *constructed* from the concrete objects by molding, building or working. The abstract models are produced by two ways: (a) by *conceptualizing* the most essential features of the concrete object(s), or (b) by ***mapping*** from some other abstract model(s). The linguistic models are produced (a) by ***symbolising*** abstract model(s) by the signs of some language and by making them visible by *representation*, or (b) by *translating* them from some other linguistic models. If the language is formal, we call the process *formalization*. The whole process of producing a model for a certain purpose is called *modelling*.

---

[2] In the literature the terms 'object type' and 'object intance' are more commonly used. We prefer here the terms that are consistent with the terminology in the semiotics.

[3] The notion of a model applies to several kinds of phenomena, e.g. things, styles or even persons (Minsky, 1965, Webster, 1980). In the literature a plethora of application-specific terms and concepts for a model have been presented. Within the limits of this paper it is not possible to discuss them.

[4] This standpoint differs from the Platonic view which considers the concepts having the existence independently from a language (Lyons, 1977).

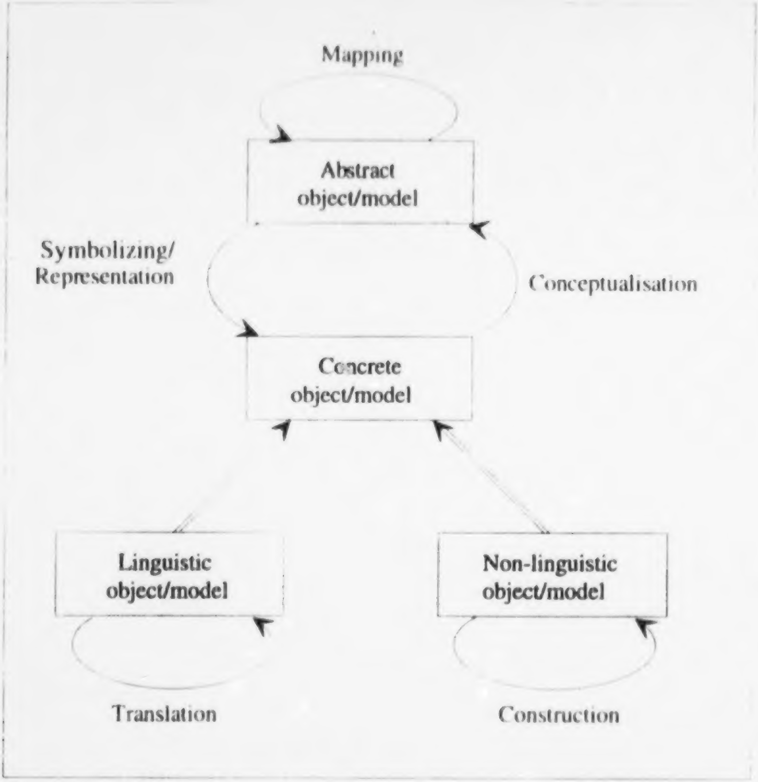

Mapping
Abstract object/model
Symbolizing/ Representation
Conceptualisation
Concrete object/model
Linguistic object/model
Non-linguistic object/model
Translation
Construction

Figure 2. Models and processes of modelling

In the following, we mainly consider the abstract models. As defined above, the (abstract) model is composed of concepts and conceptual constructs. Models can be categorized according to (1) what kinds of concepts they contain, (2) what is the primary use of the models, and (3) what is the abstraction level of their concepts. Based on the kinds of concepts, the models are divided into the structural models and the dynamic models. The *structural models* are composed of concepts that refer to static phenomena of the UoD. The structure may concern information (e.g. ER-model (Chen, 1976)), a social organization (e.g. Organisational chart (Ouchi,1978)), software (e.g. application architecture (Martin, 1982)), hardware, or other infrastructures of an organization. The *dynamic models* are composed of concepts related to the behavior or evolution of the UoD (e.g. DFD-model (Yourdon and Constantine, 1978), Action diagram (Jackson, 1983)). The concepts widely used in dynamic models are Activity, Process, Event, Trigger, and State transition (Sol and Crosslin, 1992).

Based on the primary use of a model, the models are divided into the descriptive models and the prescriptive models. A *descriptive model* is used to portray or predict the relevant features of the UoD to support the analysis of the existing reality, or to design the future reality. A *prescriptive model* is conceived as a set of normative statements which specify what is permitted, forbidden or obliged in certain situations of the UoD[5].

A *method* is a prescriptive model which governs the structure and behavior of the UoD. It gives rules for why, how, when, where and who should act. Let us consider the ER-model (Chen, 1976) as an example of a model. It is composed of concepts and constructs that refer to the existing or allowed ER-schemes. The ER-model is an abstracted description of the schemes that are results from conceptual modelling (CM). But its specifications do not concern activities, actors neither facilities of the CM. There are special CM-methods (like the one in Benyon, 1990) that prescribe how to apply the ER-model to produce an ER-schema. While the concepts of the ER-model refer to the modelled UoD, a CM-method comprises concepts that refer to the process of modelling the UoD.

Third, we apply the principle of classification to divide the models into the instance models and the type models. An *instance model* is a conceptual structure which is mainly composed of the concepts that are instances of the concepts of the other model, called a *type model*. For example, an ER-model contains concepts like Entity type, Relationship type and Attribute. The corresponding instance concepts, included in the ER-schemes, are Person, Marriage, and Age. Hence, the ER-model is a type model and an ER-schema is an instance model.

The ISD proceeds, step-by-step, sketching, specifying, elaborating, transforming, validating and verifying IS models on several levels of abstraction. It ends up with the implementation of those organisational and technical changes that have been described by the models and seen beneficial. The type models and methods used to quide the ISD work constitute an ISD methodology. An *ISD methodology* is an organized and integrated collection of philosophies, approaches, principles, models, methods, and techniques.

4. META MODEL AND METAMODELLING

Consider three levels of concepts for which it is required that the concepts on the adjacent levels are associated by the instance_of -relations. Let us call them the meta level, the type level, and the root level. Then we can define a *meta model* as a type model which describes or prescribes a model on the type level. The latter model in turn describes or prescribes phenomena on the root level.

The model on the type level can be structural or dynamic. The type of this model determines the type of the meta model in the following way. A *meta structure model* is a meta model which describes/prescribes the conceptual structures that are used or allowed in the structural models. A *meta process model* is a meta model which describes/prescribes the conceptual structures that are used or allowed in the process models.

Let us consider two examples. First, we regard a data base as an (referent) object on the root level. The corresponding ER-schema is a structural model which describes/prescribes the contents and semantics of the data base. The ER-model is the meta structure model which describes/prescribes the concepts and constructs in the ER-schemes. Second, let us regard the data processing, executed by a computer, as an object on the root level. The program is a prescriptive process model for the processing. The specification of the program language is a meta process model which describes/prscribes what kinds of processes are allowed in the programs.

---

[5] There is an important ontological difference between the models. A descriptive model should match, at a given level of abstraction, the modelled UoD. If it fails in this matching, the model is false. Contrary to that, if the matching between a prescriptive model and the UoD fails, corrective actions are required to get the UoD to fit the prescriptive model.

The selection of a root level determines what the meta model level is. Instead of a data base, we could take the reality referred by the data base as the root level resulting that the ER-schema is a meta model. Or alternatively we could regard an ER-schema as the root level. In that case, the meta model is a model which describes/prescribes the concepts and conceptual constructs of the ER-model.

The meta model is always a structural model. It means that it does not contain the concepts on how to produce a type model. The ER-model is a meta model but the instructions for producing an ER-schema are provided by a method which is a prescriptive model for conceptual modelling.

A meta model is an abstraction of the chosen features of the model(s). Those features cover the concepts, the symbols, and/or their signification relations. A *meta concept model* is a model which is an abstraction (by classification) of the concepts and conceptual structures of the model(s) on the type level. For instance, a meta concept model could be abstracted from the concepts and the conceptual structures used in the most well-known ER-models. A *meta symbol model* is a model which is an abstraction (by classification) of the symbols and formation rules of the linguistic model(s) on the type level. This kind of meta model merely manifests that certain symbols are to be used in a way which is regulated by predefined formation rules (e.g. two-dimensional graphical symbols are connected to one another by lines). A *meta signification model* is a model which is an abstraction (by classification) of the concepts, the symbols and their signification relations. By this meta model one can, for instance, state that no symbol can be used to refer to more than one concept (a uniqueness constraint).

The process of producing a meta model is called *metamodelling*. Metamodelling is a kind of modelling, but in a more restricted sense. A meta model is abstracted (by classification) from the type model(s). Another way to produce a meta model is the process of mapping from some other meta model(s). The resulting (abstract) meta model can be symbolized and represented as a linguistic meta model. A linguistic meta model can also be produced by translating it from some other linguistic meta model. The types of meta models and the processes of metamodelling are shown in Figure 3.

5. EXPLOITATION OF METAMODELLING

Metamodelling produces structured, often formal, descriptions of models, methods and methodologies, thus improving their analysis, design and implementation. Here we will first discuss the advantages of metamodelling, and then caution against some misconceptions and pitfalls in metamodelling. Below a set of applications are mentioned which may benefit from metamodelling:

(1) *Teaching, analysing and comparing models, methods and methodologies*

A meta model represented in an illustrative and concise (graphical) form can help one conceiving and structuring the symbols, the concepts, the referents and their relations within a model, a method, or a methodology. A meta model can be used as a systematic means for the conceptual analysis of consistency, coverage, clarity, and complexity of concepts and conceptual structures. To aid comparisons between divergent models, methods and methodologies, a meta model provides a uniform platform to explore

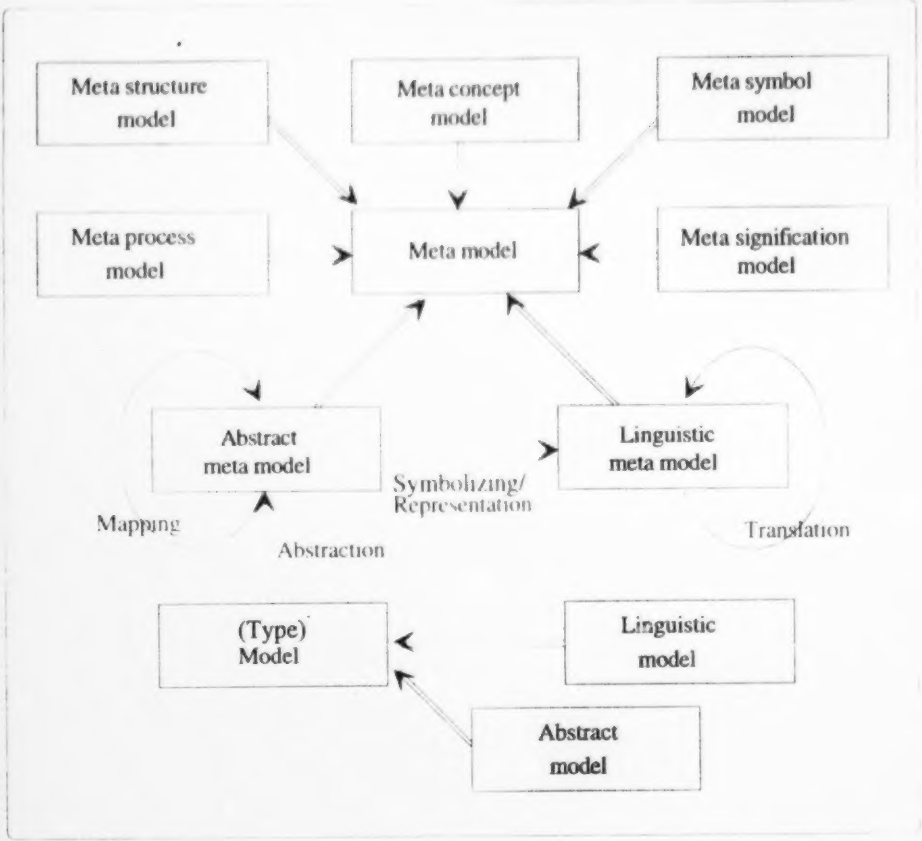

Meta structure model
Meta concept model
Meta symbol model
Meta process model
Meta model
Meta signification model
Abstract meta model
Linguistic meta model
Symbolizing/ Representation
Mapping
Abstraction
Translation
(Type) Model
Linguistic model
Abstract model

Figure 3. Meta models and processes of metamodelling

counterparts for the concepts and symbols (e.g. Goldwin, Gleeson and Gwillian, 1989). By this platform, conclusions about similarities and differences can be deepened from those presented in earlier studies (e.g. Hackathorn and Karimi, 1988, Godwin et al. 1989, Flynn and Fragoso-Diaz, 1993).

(2) *Analysing, managing and promoting the interoperability of models and methods*

An ISD-project involves a great number of development activities that are supported by a large variety of models and methods. A prerequisite for the proper support is that the models and the methods included in a methodology are interoperable (Kinnunen and Leppänen, 1994). The *structural interoperability* measures the consistency of the concepts of meta models. The *dynamic interoperability* measures how consistent the functional relationships of the concepts of dynamic models are with the structural relationships of the concepts of the meta structure models. Metamodelling can be used to assess the interoperability. Meta structure models reveal the counterparts in meta models. Dynamic models of the ISD-methods and ISD life cycle, produced according to some meta process model, manifest the phases and their decomposition into logical and temporal structures through which the ISD work is to proceed.

(3) *Transformations between the IS-models*

Occationally, it becomes necessary to transform an IS-model to apply some other type model. This is especially a case when a new ISD-approach is adopted in an organisation. There are some suggestions in the literature for carrying out transformations (e.g. Benyon-Davies, 1992). However, the transformation of an ER-diagram into an object diagram, for instance, is not merely the translation from one symbolic language to another. It calls for an in-depth analysis of concepts of the corresponding type models, to be carried out by the aid of meta models.

(4) *Reconsideration of the conceptual boundaries between models and methods*

A methodology is an artifact whose contents and structure usually are results from a more or less random evolution. Existing categorizations of concepts into the models and the inclusion of processes into the methods may turn out to be unsuitable for perceiving the relevant phenomena of the UoD or for accomplishing necessary changes in the UoD. The meta models enable the reconsideration of the conceptual boundaries between the models and the methods.

(5) *Development of models and methods*

A meta structure model can be used to consider the pros and cons of making a type model semantically more rich. Let us consider an ER-model as an example. The original ER-model (Chen, 1976) has been commonly extended with refined concepts and constructs (e.g. Elmasri and Wuu, 1990, Theodoulis et al., 1991). Extending the ER-model by new type concepts (e.g. by temporal concepts) makes it unnecessary. to. define explicitly special features of the UoD by an instance model. By metamodelling the conceptual relations between the existing and new type concepts can be defined and analyzed. To increase the flexibility a generic type model can be equipped with generalization hierarchies. For instance, a new adaptable ER-model can be developed which contains three main concepts: Entity type, Relationship type and Atrribute. For those applications for which dynamic features are intrinsic, the concept of Entity type is specialized into the concepts of Static entity type and Point of time. To support the temporal modelling, a set of assertion types, based on some temporal logic, is also provided. A meta structure model can be used to ensure that different options for extending a type model concepts do not complicate the conceptual modelling in ordinary applications

(6) *Customization of models, methods and methodologies*

No model, method or methodology as such is usable for an ISD project. Some changes, deduced from the contingencies of an application area, staff, resources available, etc, are always required. Meta models help generating fertile versions of models and methods and assessing the affects of these changes upon the other components of a methodology. A dynamic model of a methodology life cycle, produced according to a meta process model, aids the project management to decide on the phase structure, milestones and deliverables (Heym and Österle, 1992, Hahn, Jarke and Rose, 1991).

(7) *Development of meta-CASE environments*

In recent years a wide range of CASE tools have been developed to make ISD work more efficient. By these days CASE tools have been bounded to some specific models or methods causing difficulties in their implementation in organisations which are used to different models and methods. Metamodelling can be deployed to construct metaCASE environments which offer the support for a set of models and methods as well as the facilities for customising them (e.g. Bergsten et al., 1989, Smolander et al., 1991).

The concepts related to the meta models and metamodelling are quite frequently used without grounding them on a sound theoretical basis (Saeki et al., 1993, van Slooten and Brinkkemper, 1994). This is one reason for an inconsistent use of even the basic concepts. For example, Brinkkemper, one of the most notable pioneers in metamodelling, defines the meta-activity model to mean a generic description of a procedure embedded in a method (Brinkkemper, 1990). This however is not a proper use of the term 'meta'. A meta-activity model does not model any type model but development processes that follow the guidelines of a method.

Another misconception is related to the concept of a meta-methodology. DeMargo (1984) was one of the first in deploying that concept. But he defined it to mean "a tool kit of well documented activity method statements". He suggested that a meta-methodology should be customised to serve the needs of a certain development project by selecting a subset of methods. This conception does not reflect the instance_of relation but something which is related to selection and tailoring. Recently several suggestions for the frameworks have been introduced to cover all the essentials of ISD-methodologies. They are most commonly called by the terms 'frame of reference' and 'reference model' (Essink, 1988, Iivari, 1990, Pulst et al., 1990). They contain meta models but also descriptions which are not on a meta level. By the latter we mean the descriptions of fundamental philosophies, approaches, principles and intended application areas of a methodology.

The third issue discussed here concerns the principles of producing a meta model. Because a meta model is composed of type concepts, we concentrate on a way of defining a type concept. Generally speaking, a concept can be defined extensionally and intentionally. When defined extensionally, a type concept is constructed from characteristics of the corresponding instance concepts. The most common ways to construct a definition of a concept from the other concepts is to apply the operators of intentional union or intersection (Bunge, 1977) to the sets of characteristics. But it is here important to notice that only the operator of intersection is allowed when constructing a type concept. When defined intensionally, characteristics of a type concept are derived using a sound theoretical basis. This means that the defining process proceeds independently from the existing instance concepts. Not until a definition of a type concept is completed, its relations to the existing instance concepts are examined.

The forth issue is concerned with the superficiality that is characteristic of some arguments behind metamodelling. The conceptual background of two type models can differ so much from one another that the pursuit of defining explicit relations between their concepts calls for an in-depth conceptual analysis. This is especially true for the models stemming from different approaches (cf. the ER-approach and the object-oriented approach) but also for the models of the same approach (for example, the concept of an object may be used quite differently in the models of two object-oriented methodologies). One should always remember that a model described by the meta concepts is not any more the same model. Metamodelling is an abstraction process which reveals the essentials of the concepts but at the same time hides, intentionally or unintentionally, other characteristics of the concepts. Often those characteristics might be of high importance.

Finally, it should never be forgotten that metamodelling is just a means of making a conceptual analysis of models, methods and methodologies. From the viewpoint of practical work, the most important thing is that a model, a method or a methodology is applicable and acceptable. Although the consistency, clarity, coverage and simplicity of the concepts and constructs are significant ingredients of applicability and acceptability, still more important are those experiences which have been gained when applying a model, a method or a methodology in different kinds of development projects. This kind of method validation is not without problems, as Fitzerald (1991) and Grant et al. (1992) note, but an absolute prerequisite for the ultimate assessments.

REFERENCES:

Arnauld, A., (1964), "The Art of Thinking (Port-Royal Logic)", Bobbs-Merrill, New York.

Benyon, D., (1990), "Information and Data Modelling", Blackwell Scientific Publications, Oxford.

Benyon-Davies, P., (1992), Entity models to object models: object-oriented analysis and database design, *Information and Software Technology*, Vol. 34, No. 4, pp. 255-262.

Bergsten, P., Bubenko, J., Dahl, R., Gustafsson, M. and Johansson, L.-Å., (1989), "RAMATIC - a CASE Shell for Implementation of Specific CASE Tools", TEMPORA T6.1, SISU, Stockholm, Sweden.

Brinkkemper, S., (1990), "Formalisation of Information Systems Modelling", Ph. D. Thesis, University of Nijmegen, The Netherlands.

Brinkkemper, S., Geurts, M., van der Kamp, I. and Acohen, J., (1989), On a formal approach to the methodology of information planning, in "Proceedings of the First Conf. on Information Ssytems" (ed. Maes, R.), Amersfoort.

Bunge, M., (1967), "Scientific Research I. The Research for System", Springer-Verlag, New York.

Bunge, M., (1977), "Treatise on Basic Philosophy (Volume 3): Ontology I, The Furniture of the World", D. Reidel Pub. Co., Boston.

Chen, P., (1976), The entity-relationship model - toward a unified view of data, *Trans. on Database Systems*, Vol. 1, No. 1, pp. 9-36.

DeMarco, T., (1984), A meta-methodology for systems development, in "International Workshop on Models and Languages for Software Specification and Design", Orlando, Florida, 1-6.

Elmasri, R. and Wuu, G., (1990), A temporal model and query language for ER databases, in "Proc. of the International Conf. on Data Engineering ", IEEE, 76-83.

Essink, L., (1988), A conceptual framework for information systems development methodologies, in "Information Technology for Organisational Systems" (eds. Bullinger H.-J. et al.), North-Holland, Amsterdam, pp. 354-362.

Fitzgerald, G., (1991), Validating new information systems techniques: a retrospective analysis, in "Information Systems Research: Contemporary Approaches and Emergent Tranditions" (eds. Nissen H.-E., Klein, H. and Hirschheim, R.), North-Holland, Amsterdam, pp. 657-672.

Flynn, D. and Fragoso-Diaz, O., (1993), Conceptual EUROModelling: how do SSADM and MERISE compare?, *European Journal of Information Systems*, Vol. 2, No. 3, pp. 169-183.

Godwin, A., Gleeson, J. and Gwillian, D., (1989), An assesoment of the IDEF notation as descriptive tools, *Information Systems*, Vol. 14, No. 1, pp. 13-28.

Godwin, A., Gore, M. and Salt, D., (1989), A comparison of JSD and DFD as descriptive tools, *The Computer Journal*, Vol. 32, No. 3, 202-211.

Grant, D., Ngwenyama, O. and Klein, H., (1992), "Validating ISD Methodologies Within the Organizational Context: an Action Research Case Study", Working paper series, Binghampton, State University of New York, 92-125.

Hackathorn, R. and Karimi, J., (1988), A framework for comparing information engineering methods, *MIS Quarterly*, June, pp. 203-220.

Hahn, U., Jarke, M. and Rose, T., (1991), Teamwork support in knowledge-based information systems environment, *IEEE Trans. on Software Engineering,* Vol. 17, No. 5, pp. 467-481.

Hautamaki, A., (1986), Points of view and their logical analysis, *Acta Philosophica Fennica,* Vol. 41, Helsinki.

Heym. M., Osterle, H. (1992), A reference model for information systems development, in "The Impact of Computer Supported Technologies on Information Systems Development" (eds. Kendall, K., Lyytinen, K., De Gross, J.,), North-Holland, Amsterdam, 215-240.

Iivari, J., (1990), Hierarchical spiral model for information system and software development. Part 1: theoretical background, *Information and Software Technology*, Vol. 32., No. 6, pp. 386-399.

Iivari, J., (1990), Hierarchical spiral model for information system and software development. Part 2: design process, *Information and Software Technology*, Vol. 32., No. 7, pp. 450-458.

Jackson, M. A., (1983), "System Development", Prentice-Hall, Englewood Cliffs, New Jersey.

Kinnunen, K. and Leppanen, M., (1994), O/A matrix and a technique for methodology engineering, in "Proceedings of the ISD'94-Conference", Bled, Slovenia.

Lindgren, P. (ed.), (1990) "A Framework of Information Systems Concepts", Report of the IFIP WG 8.1 Task Group FRISCO.

Lyons, J., (1977), "Semantics", Cambridge University Press, Cambridge.

Martin, J., (1982), "Strategic Data-Planning Methodologies", Prentice-Hall, New Jersey.

Meersman, R., Shi, Z. and Kung, C.-H. (eds.), (1990), "Artificial Intelligence in Database and Information Systems (DS-3)", Proc. of the IFIP TC2/TC8/WG2.6/WG8.1 Working Conf., North-Holland, Amsterdam.

Minsky, M., (1965), Models, minds, machines, in "Proceedings of AFIPS Conference", AFIPS Press, Montcale, New Jersey.

Morris, C., (1964), "Signs, Languages, and Behaviour", Prentice-Hall, New York.

Nassi, I. and Shneiderman, B.,(1973), Flowchart techniques for structured programming,*ACM SIGPLAN Notices,* Vol. 8, No. 8, pp. 12-26.

Ogden, C., Richards, I., (1923), "The Meaning of Meaning", Kegan Paul, London.

Olle, T., Sol, H. and Verrijn-Stuart, A. (eds), (1982), "Information Systems Design Methodologies: a Comparative Review", North-Holland, Amsterdam.

Ouchi, W. G., (1979), A conceptual framework for the design of organizational control mechanisms, *Management Science,* Vol. 25, No. 9, pp. 833-848.

Pulst, E. et al., (1990), "HECTOR's Framework of Reference", Esprit Project 2082.

Saeki, M., Iguchi, K., Wen-yin, K. and Shinohara, M., (1993), A meta-model for representing software specification & design methods, in "IFIP Transactions A30: Information System Development Process" (eds. Prakash, N., Rolland, C. and Pernici, B.), North-Holland, Amsterdam, pp. 149-166.

van Slooten, K. and Brinkkemper, S., (1993), A method engineering approach to information systems development, in "IFIP Transactions A30: Information System Development Process" (eds. Prakash, N., Rolland, C. and Pernici, B.), North-Holland, Amsterdam, pp. 167-188.

Smolander, K., Lyytinen, K., Tahvanainen, V.-P. and Marttiin, P., MetaEdit - a flexible graphical environment for methodology modelling, in "Advanced Information Systems Engineering, Proceedings of the CAiSE*91 Conf" (eds. Andersen, R., Bubenko, J. and Sölvberg, A.), Springer-Verlag, New York, 168-193.

Sol, H. G. and Crosslin, R. L. (eds.), (1992), "Dynamic Modelling of Information Systems, II", North-Holland, Amsterdam.

Theodoulidis, C., Loucopoulos, P. and Wangler, B., (1991), A conceptual modelling formalism for temporal database applications, *Information Systems,* Vol. 16, No. 4, 401-416.

Webster's New World Dictionary, (1980), 2. Edition, Simon and Schuster, New York.

Yourdon, E. and Constantine, L., (1978), "Structured Design", Yourdon Press, New York.

System Development

Linguistically Based Information Systems Development - A Constructive Approach

Erich Ortner

Universität Konstanz, Fakultät für Verwaltungswissenschaft, Postfach 5560,
78434 Konstanz 1, Germany

Abstract:

The paper introduces a method for developing software systems from expert languages. The initial developing steps consist in the collection of relevant propositions concerning facts in an application area, and the reconstruction of the expert concepts which these propositions are based upon. This yields to a consistent collection of propositions which all further development of results can systematically be derived from. The method is outlined for software engineering as a whole. In support of this approach the connection between expert concept systems and the facts in a section of reality which was described by means of them is explained. Using data modelling as an example, we will demonstrate how relevant propositions can serve as a base for developing step by step an organization-wide data model. It will be shown to which extent the results of this method are relevant for the use of software systems and for the organization's management.

1. INTRODUCTION

Linguistic criticism as a starting point for methodical language reconstruction leading from pretheoretic practice to terminologies supporting practice: that is the program of constructive philosophy (9, 10). Based on this (11), the method allows to construct and operate application systems in computer-supported information processing which are adequate to users' needs. Linguistic criticism as a construction method for software is to be understood here as follows: When we use language, we depict facts. We express wishes, make assertions, ask questions or give instructions. Such speech acts also take place in information processing application areas on the level of natural language (in an expert language). From this starting point software engineering projects can proceed by analyzing systematically both content and structure of linguistic expressions (sentences/phrases) to reconstruct the concepts which underlie the utterances. In this process, the situations out of the application area are described more precisely, and the expert concepts which support them are consistently defined. Thus one moves from a often vague, unmediated use of language to a clarified and regular use of (expert) languages. Software systems can be developed from the collection of relevant propositions with clarified expert concepts more effectively and with a higher degree of acceptability for the user.

This paper is divided into three sections: The first section contains a typical problem situation and its possible solution by means of linguistic criticism, as it may be found initially in software engineering projects. We will point out, how expert concept systems and the objects or facts of an information processing application area subsumed under those concepts are connected. It will show that expert solutions to construction tasks are always given in conceptual form.

In the second section the application of this general principle for solving construction tasks is demonstrated in a sub-area of software engineering, the area of data modelling (4), (2). Here we can refer to practical experiences which have already been made with this approach (12). We will demonstrate the step-by-step development of an organization-wide data model, primarily by reconstructing the concepts for information objects and their characteristics. In the course of this reconstruction process vagueness, fuzziness and incongruety of and between the concepts need to be eliminated so that the expert concepts, now free of interference, can be used uniformly by everyone in the organization. Only on the basis of well defined expert concepts in an organization integrated information processing becomes possible.

In the third section a scientific program is introduced which aims at developing software systems from the linguistically represented knowledge to information processing application areas. In this process the method based on linguistic criticism clearly not only shows the way to the development of adequate software solutions, but the well-defined concepts create the necessary conditions for the effective use of the systems. Further, it enables the management to carry out information processing in an integrated manner.

2. CONSTRUCTING WITH CONCEPTS

When we more closely analyse a more or less defined application area, for example the accounts or personel branch of an organization, we find an area which is already organized. The structure is the result of an introduced system of expert concepts such that particular concepts of the concept system are assigned to individual objects in the application area, and whole sections of the concept system are assigned to events, situations or facts within the application area. The concept system is in short the expert theory of the application area in question, and the facts verified document its practice.

The sense of the theories is to stabilise and support operational practice. They are, to use the words of the mathematician and philosopher Pascal, "extreme stylisations of what we already have done in normal life".

The simple situations: Miller is an employee, D 37 is a department, Miller works in D 37, can therefore be verified and classified by us, because: what is to be understood by EMPLOYEE, when one can speak of a DEPARTMENT, what meaning the relationship WORKS IN has in this context, is conceptually fixed.

The concepts EMPLOYEE (x), DEPARTMENT (y) and WORKS IN (x,y) as propositional functions forming the concept system (connections) through the argument variables x and y, are decisive for the possibility of verifying the above situations.

True situations are called facts. Therefore the argument variables of the concepts or proposition functions are replaced by the names of those objects, for which the assertions under consideration can be verified, here "Miller" for "x" and "D 37" for "y". The named objects (Miller, D 37) fall within the concepts - they belong to their extension - when the assertions put forward are true. The objects do not fall within the concepts when the assertions are false. This operational model for concepts was first developed by Frege (7) and on this basis the (formal) logic was newly conceived.

Thus a concept system which prevails in an information processing application area represents the knowledge which the persons who need to perform their work efficiently in this area and to make reliable decisions must command. Only on the basis of a common concept system effective communication becomes possible. The goal of a software solution is to support user activities and also decision-making in the application areas. It is thus natural that the software should also be conceived on the basis of a system of expert concepts which is part of an application area. Software engineering consists in its first developing step in the reconstruction of those concepts which are common to both the desired software solution and its subsequent users. This first step is called conceptual system design, or in short conceptual design. Technological issues thus stay in the background. The reconstruction and clarification of the users (expert) terminology in the application area to be supported by the software are placed in the foreground.

3. DATA MODELLING IN ORGANIZATION AS AN EXAMPLE

Data form a part of a conceptual system design. The data are the "raw material" from which, by means of processing in software systems, relevant information emerges for action-takers and decision-makers. Furthermore, the separate organization of data, in particular data modelling, is an important step towards the integration of software systems in organizations.

The 3-scheme architecture (1) according to ANSI/SPARC (American National Standards Institute/Standard Planning and Requirement Committee) provides the structural framework for the achievement of this integration by means of data. The three scheme fields external, internal and conceptual contain propositions *about* the organization of the data. They are thus to be assigned to the data category "metadata". Internal schemes (IS) specify how the storage of data on workplace systems or on the central computer is carried out. The external schemes (ES) define the form (composition) in which the data are requested by the individual applications (programs). The conceptual scheme (CS) establishes the semantics (concepts) of an organization's data which are common to all usage forms (ES). Conceptual schemes are also known as organizational data models or concept schemes for the data resources of an organization to define the uniform specification of user concepts for the data from the application areas. Because of this they are characterised above all by the "representation" of the expert (user) concepts and their relationships (Figure 1). The conceptual schema appears as a network of expert concepts each of which is implemented in a way which extends across applications and is binding for all in the organization. The expert concepts are called object types in the data model (Figure 1), and inclusive (△), aggregative (Δ) or connective (◇) concept connections can be reconstructed.

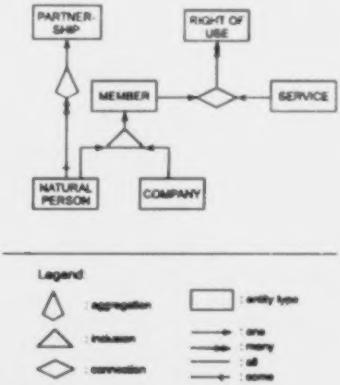


Figure 1: Conceptual part-scheme (an example)

We speak of inclusive concept relationships, *inclusions* for short, when an object area is seen from various points of view, i.e. concepts. Linguistic norms of words exist in the user area, leading to a class hierarchy in the sense of a generalisation or specialisation of perspectives of an object area (14). The concepts NATURAL PERSON and COMPANY (legal person) define the same object area more specifically than in our example the concept MEMBER (Figure 1). *Aggregations* lead to the grouping together of objects of a concept under the objects of another concept. Normally the object areas here are different. So, for example, (Figure 1) objects of the concept NATURAL PERSON are grouped together under objects of the concept PARTNERSHIP. Those (natural) persons who as members of an organization form themselves into partnerships. They are then conceptually assigned in the data model by means of the reconstructed relational type "aggregation". In (5), amongst others, this specific aggregation type is called "cover aggregation". *Connections*, finally - the third type of relationship - lead to the introduction of a new concept in a data model. Here relationships which are reconstructed between objects of particular concepts become objects themselves and are grouped together under a new concept. Every use of a SERVICE by a

MEMBER is explicitly listed as a newly formed object under the new concept RIGHT OF USE (Figure 3). In this example the verification that a MEMBER can use several SERVICES, whereas a SERVICE can be used by several MEMBERS precedes the new concept formation RIGHT OF USE. Since the aim is to record specific information about individual exercises of services, the new concept RIGHT OF USE is adopted in the data model.
The relationships represented in a data model by means of a relational symbol - triangle (△), trapezoid (⏢) or diamond (◇) - and their object types, are called relational complexes (Figure 1). Note that, in general, n-ary relationships between object types cannot be reduced (without information loss) into binary relationships.
Thus data modelling is that part of the conceptual design of a software solution where the objects of the information processing, their relationships and also the characteristics (attributes) of objects and relationships are reconstructed on the conceptual level consistent with expert (user) practice and laid down in a binding manner for all in the organization. In addition, integrity constraints also belong to a complete data model design, that is, further propositions about the consistency of the data resources of an enterprise.
The central question is now how to arrive at an enterprise-wide data model. It is not possible to model an enterprise-wide data model in an organization in a single step which will remain stable forever. Data modelling is a permanent task in enterprises, which has to take place at various organizational levels and in a variety of situations. It is used not only in the construction of new systems, but also in strategic information planning and in the reconstruction and servicing of the application systems which have been put in place.
Facing this question, an approach which is based on linguistic criticism may reveal itself as a valid alternative to strategies following either of two main points of view which are mostly put forward in computer science, one of which we may characterize as the "naive" or "empirical" position, the other as the "dogmatic" position. The *naive* point of view is adopted when it is assumed that a very simple representational formalism (for example, a graphic notation or diagram technique, cf. Figure 1) which one applies as means of communication between the application area and the development leads to an appropriate representation of complex concepts and concept relationships in the expert branches. Adequate modelling results are seldom achieved by such representational means. The specific reconstruction process takes place at best in an "intuitive" manner, or does not take place at all. At some stage the users agree to the graphic representation just for convenience. The common goal, user concepts capable of consensus throughout the organization, is, however, seldom reached in this way.
On the other hand one adopts a *dogmatic* position when an expressive representational formalism which is available for the implementation of the systems (for example, PROLOG, relational data model) is already employed at the stage of modelling the conceptual solutions. Very often semantically distorted representations of the facts relevant to the development occur in the modelling due to the "claim to unconditional recognition" of representational

formalism's formal requirements (for example, 1st normal form of the relational model) The direct participation of the user is no longer possible here. The chosen representational formalism is not a suitable means of communication between development and the application area. The results achieved are comparable with those which are yielded when taking the naive point of view as a starting point. The "conceptual solutions" specified at first appear to fulfill the users' needs. Following the introduction of the system, however, they often prove to be inappropriate, and are not accepted by the users. The decisive argument against both points of view is this - representational formalism is not relevant at all yet in this phase of the system development. Even if we could program in natural language, we would not be released from the task of reconstructing the user concepts - the primary achievement of conceptual design.

As stated above, there is a strategy which differs in fundamental aspects from both naive and dogmatic points of view: the approach based on *linguistic criticism*. First of all, the prevailing expert concept system in an application area is systematically reconstructed step by step and without circles, proceeding from relevant (natural language) propositions. Lack of clarity, contradictions or flaws in and between the concepts are to be eliminated. On this basis the best conceptual solution can be conceived and subsequently implemented on a computer system. What is still explained in a somewhat "abstract" manner can be very simply carried out using the method. This will be demonstrated by means of an example from the organization DATEV (Data processing organization of the tax consultants' profession in the Federal Republic of Germany.)

The first results of the modelling work are indeed not boxes, circles or relationships in a diagram technique, but rather natural language proposition in the form:

- MEMBERS OF DATEV can be NATURAL PERSONS or COMPANIES.
- Only NATURAL PERSONS can form themselves into a PARTNERSHIP.
- DATEV recognizes PARTNERSHIPS as users of their SERVICES only when and if all PARTNERS are also MEMBERS of DATEV.

etc.

Such propositions remain unclear, incorrect or contradictory, because the underlying concepts are unclear, incorrect or contradictory.

- How do NATURAL PERSONS differ from COMPANIES (in the DATEV context?)
- What are PARTNERS?
- Can PARTNER AND PARTNERSHIPS be used synonymously?
- What are the SERVICES offered by this organization?
- How is DATEV defined?

The second step is the clarification and reconstruction of those expert concepts which underpin these propositions.

It is important now to turn to the concepts in detail, to investigate them, to clarify and uniformly regulate their use in the application areas. In doing this attention has to be paid to the use of synonyms and homonyms. Equipollences, that is, concepts which have the same extension but different intensions, such as the concepts GOODS ACCOUNT and STOCK INVENTORY, have to be detected. The intension of vague concepts has to be precised (intensionally) so that the extension is always clear. False designations for concepts which suggest another meaning than that have to be replaced by better concept words.

The result of step one (collection of relevant propositions) and two (clarification and reconstruction of the expert concepts) is a collection of propositions which needs to be updated, completed and adjusted to the most recent developmental status on an ongoing basis.

All further results of the conceptual design (here object types, relational complexes, attributes and integrity constraints) can be systematically derived from this stock of propositions. Therefore, the propositions must be classified appropriately and formed into groups. In doing this one considers first of all the structure and the connecting words in the propositions. For database applications it is useful to form groups of:

- propositions about objects and object characteristics,
- propositions about operations with the objects (their data),
- propositions about events which generate operations in the data stores, and
- propositions about integrity constraints

Multiple assignments are possible. The stock of propositions pre-structured in a first step is prepared for the transition to object types and relationships of a data model by further grouping of the propositions (here the propositions about objects and object characteristics as well as statements about integrity constraints). At this stage also the connecting words or particles in the propositions provide orientation for the transition to object type constructions.

Propositions of the following form (Figure 2) establish *inclusive* conceptual relationships between the object types:

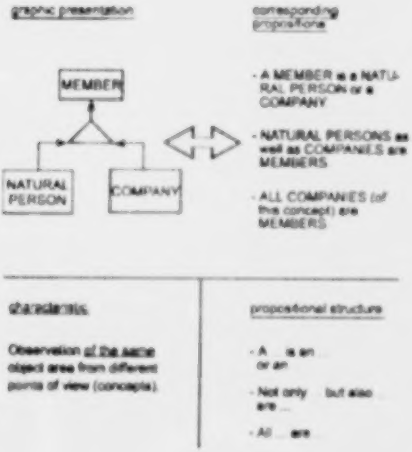
- A MEMBER (of DATEV) is a NATURAL PERSON or a COMPANY.
- Not only NATURAL PERSONS but also COMPANIES are (DATEV) MEMBERS.
- All COMPANIES (of this concept) are (DATEV) MEMBERS.

It is the same object area, that is DATEV MEMBERS, which is seen from various points of view, here from the point of view of all MEMBERS, from the point of view of the NATURAL PERSONS, and from the point of view of the COMPANIES. The connecting words "or", "not only but also" in the propositions show the conceptual structure to be an inclusion.

Propositions of form (Figure 3):

- A PARTNERSHIP consists of several NATURAL PERSONS.
- A NATURAL PERSON belongs (if at all) to a PARTNERSHIP.
- There are NATURAL PERSONS who do not belong to a PARTNERSHIP.

are based on the *aggregative* relationships between the concepts involved.

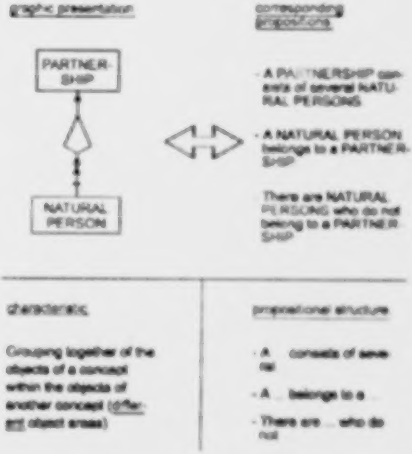

graphic presentation
corresponding propositions
MEMBER
NATURAL PERSON
COMPANY
characteristic
propositional structure

- A MEMBER is a NATURAL PERSON or a COMPANY
- NATURAL PERSONS as well as COMPANIES are MEMBERS
- ALL COMPANIES (of this concept) are MEMBERS

Observation of the same object area from different points of view (concepts).

- A ... is an ... or an ...
- Not only ... but also ... are ...
- All ... are ...

Figure 2: Inclusion


graphic presentation
corresponding propositions
PARTNERSHIP
NATURAL PERSON
characteristic
propositional structure

- A PARTNERSHIP consists of several NATURAL PERSONS
- A NATURAL PERSON belongs to a PARTNERSHIP
- There are NATURAL PERSONS who do not belong to a PARTNERSHIP

Grouping together of the objects of a concept within the objects of another concept (different object areas)

- A ... consists of several ...
- A ... belongs to a ...
- There are ... who do not

Figure 3: Aggregation

Here the objects of a concept (NATURAL PERSONS) are grouped together under the objects of another concept (PARTNERSHIP). Aggregative conceptual relationships are characterised by connecting words in the propositions such as "belongs to", "consists of", "part of".

And finally, propositions in the form (Figure 4):

- A MEMBER can use several SERVICES.
- A SERVICE can be used by several MEMBERS.
- The option for a MEMBER to use a SERVICE is called a RIGHT OF USE.

lead to the introduction of *connective* conceptual relationships. The connections (relationships) verified between the object areas of the concepts MEMBER and SERVICE become objects themselves and are grouped together under the newly formed concept RIGHT OF USE. In that situation a proposition always appears in the proposition collection, here the proposition

- The option of using a service by a MEMBER is called RIGHT OF USE,

which explicitly introduces the newly formed concept RIGHT OF USE.

BEST COPY AVAILABLE

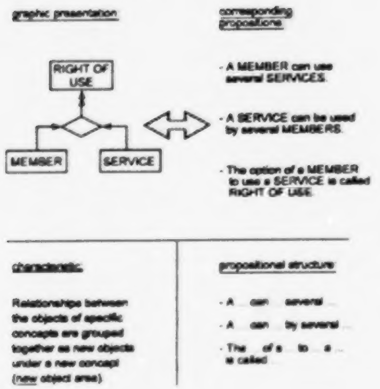

graphic presentation
corresponding propositions
RIGHT OF USE
MEMBER
SERVICE
- A MEMBER can use several SERVICES.
- A SERVICE can be used by several MEMBERS.
- The option of a MEMBER to use a SERVICE is called RIGHT OF USE.
characteristic:
Relationships between the objects of specific concepts are grouped together as new objects under a new concept (new object area).
propositional structure:
- A ... can ... several ...
- A ... can ... by several ...
- The ... of a ... to ... a ... is called ...

Figure 4: Connection

The "data model" presented is much too simple to give an idea of the complexity of the aim and the effort to be expended. Thus it cannot be described as an "enterprise-wide data model". At an advanced stage of development a data model for a medium-sized organization (enterprise) consists of several hundred object types and relationships. In addition, the object types only define a part of the data model, namely the broad or macro structure. Every object type shows up to 40 attributes, which define the characteristics of the information objects belonging to these object types. So called integrity constraints - further propositions about the data resources of an organization- are added to ensure consistency. At the level of attributes and integrity constraints an organizational data model can no longer be managed without technical means (3). For this software tools, so-called dictionary systems have to be used. In addition, organizational rules in relation to the construction and sequence of the data modelling work have to be made to secure co-ordinated development of the data model in an organization permanently (12).

4. A PROGRAM FOR LINGUISTICALLY BASED SOFTWARE ENGINEERING

This part contains the program of designing software systems in general - not only the data, but also the processing part (for example the programs)- by means of linguistic criticism (Figure 5).

A section of reality in expert language or colloquial language needs to be modelled. We proceed from real work situations with a regulated sequence of action and communication governed by expert language in an application area of information processing (section of reality) and systematically reconstruct the conceptual basis of this procedure. The goal initially consists in the ortholinguistic normalisation and stabilisation of the language situation which is achieved by reconstruction of the underlying expert concepts.

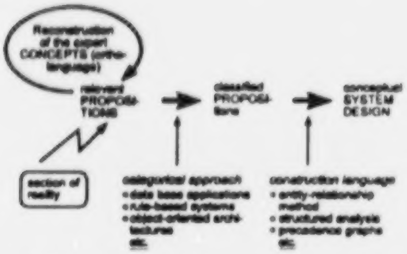

Reconstruction of the expert CONCEPTS (ortho-language)
relevant PROPOSITIONS
classified PROPOSITIONS
conceptual SYSTEM DESIGN
section of reality
categorical approach
• data base applications
• rule-based systems
• object-oriented architectures
etc.
construction language
• entity-relationship method
• structured analysis
• precedence graphs
etc.

Figure 5: Linguistic criticism as a method for conceptual system design

Ortholanguage is a terminus from constructive scientific theory (10) in which every word belonging to the stock of an expert language is first of all explicitly clarified and conclusively defined (reconstructed). In this reconstruction process the words attain an "ortho" status, that is, "enter a conforming position". The language which is constructed in this way is called ortholanguage.

We follow this procedure of constructivism at the point in time of conceptual system design. We do not aim at the new definition of the concepts of an application area, but we rather reconstruct them together with their users by means of linguistic criticism, in rendering them precise and uniform. On this basis software solutions can be developed in the further phases of the system design with a greater degree of acceptability for the users. We begin with the collection of relevant propositions about conditions in the relevant application area. We obtain the propositions by discussions with the employees at their work and by observation of the work. The contemporary collaboration of developmental engineers in the application area can be helpful. In addition one might study the expert literature, internal papers, records and other documentary material in order to analyse the facts. The result is a more or less complete stock of relevant propositions, which may still be unclear, incorrect or contradictory, if the concepts underlying them are unclear, incorrect or contradictory. Here the second developing step (using the same methods: interviews, study expert literature, etc.), the intensive consensual reconstruction of the standard expert concepts, leads to the elimination of linguistic defects and to an internally consistent stock of propositions. At this point to get to a classified stock of propositions one has to agree on fundamental principles: One has to chose a solution principle, also known as "categorical approach" suited to the task at hand.

Thus, the further developing route (Figure 5) and the translation of the classified propositions into specific design languages (construction languages) is prepared. The translation of quasi-colloquial language into a graphic notation is based on considerations of efficiency. In this way one can represent the results of the conceptual design with the relevant characteristics for the following phase of the software development more succinctly and precisely.

This conversion was demonstrated using the example of data modelling. Appropriate conversion forms need to be developed for other types of results in conceptual system design such as business processes, communication relationships, rule bases, etc. Before developing the definition of the characteristics of the system it is crucial to reconstruct the expert concepts of an application area. For this reason the program (Figure 5) suggested here is called "linguistic criticism as a method for conceptual systen design".

5. CONCLUDING REMARKS

Constructive solutions for the tasks at hand - whether in software engineering or in other engineering disciplines - are in the beginning in the form of language products, that is, they are the expression of a developed concept system which forms the solution to the problem. Special about software solutions is that they are, per se, language products, which can be used by means of computers. Here reconstruction of the "common" conceptual basis of computer applications on the one hand and their users on the other hand guarantees the effectiveness of the computer-supported information and communication process.
Linguistic criticism was established in this process as an effective means for the development of adequate computer programs. The results we have from numerous data modelling projects (12) provide clear evidence of this. A *"linguistic turn"* in software engineering will however lead to basic changes, above all in the early and late stages of system development. The rules need to be determined according to which knowledge captured in natural language can be reconstructed in application areas and the degree to which it is necessary to regulate the use of language in the application areas in using software systems. This path cannot be followed before we have achieved the complete capacity to critically reconstruct knowledge captured in expert language in the application areas. Besides data it is also necessary to conceptually reconstruct programs. We have not yet reached this stage (16). It is necessary to elucidate this unclear situation through further development of the method - focussing on the potential of constructive scientific theory (9), (10) to provide solutions. It´s likely that these steps will soon be achieved.

Literature

(1) **ANSI/X3/SPARC:** Study Group on Data Base Management, Systems-Interim-Report, in: Bulletin of ACM SIGMOD, 7 (1975) 2.

(2) **Batini, C.; Ceri, S.; Navathe, S.B.:** Conceptual Database Design, An Entity-Relationship Approach, The Benjamin/Cummings Publishing Company, Inc., Redwood City etc. 1992.

(3) **Ceri, S. (ed.):** Methodology and Tools for Database Design. Elsevier Science, North-Holland 1983.

(4) **Chen, P.P.:** The Entity-Relationship Model: A Basis for the Enterprise View of Data, Proceedings IFIPS NCC 46, No. 46, 1977, pp. 76-84.

(5) **Codd, E.F.:** Extending the Database Relational Model to Capture More Meaning, ACM Transactions on Database Systems, Vol. 4, No. 4, December 1979, pp 3978-434.

(6) **Denert, E.:** Software-Engineering, Methodische Projektabwicklung, Springer-Verlag, Berlin etc. 1991.

(7) **Frege, G.:** Funktion, Begriff, Bedeutung - Funf logische Studien, Patzig, G. (Hrsg.), Kleine Vandenhoeck-Reihe 1144, Gottingen 1975.

(8) **Jalote, P.:** An Integrated Approach to Software Engineering, Springer-Verlag, Berlin etc. 1991.

(9) **Lorenzen, P.:** Normative Logic and Ethics, 2nd annotated edition, Bibliographisches Institut AG, Zurich 1984.

(10) **Lorenzen, P.:** Constructive Philosophy, Translated by K.R. Pavlovic, The University of Massachusetts Press, Amherst 1987.

(11) **Ortner, E.:** Aspekte einer Konstruktionssprache fur den Datenbankentwurf, S. Toeche-Mittler Verlag, Darmstadt 1983.

(12) **Ortner, E.; Söllner, B.:** Semantische Datenmodellierung nach der Objekttypenmethode, in: Informatik-Spektrum, 12 (1989) 1, S. 31-42.

(13) **Reimer, U.:** Einfuhrung in die Wissensreprasentation, Reihe: Leitfaden der angewandten Informatik, B.G. Teubner, Stuttgart 1991.

(14) **Smith, J.M.; Smith, D.C.P.:** Database Abstractions: Aggregation and Generalization, ACM Transactions on Database Sytems, Vol. 2, No. 2, June 1977, pp 105-133.

(15) **Sommerville, I.:** Software Engineering, 2nd Ed. Wokingham 1985.

(16) **Wedekind, H.:** Objektorientierte Schema-Entwicklung. Ein kategorialer Ansatz fur Datenbanken und Programmierung, B.I. - Wissenschaftsverlag, Mannheim/Leipzig/Wien/Zurich 1992.

(17) **Wilson, M.C.:** The Measurement of Usability, in: Entity Relationship Approach to Systems Analysis and Design, Chen, P.P. (Ed.), North Holland, 1980, pp 87-113.

DATA HANDLING: A PERSPECTIVE OF SIGN, INFORMATION AND MEANING

Jun-Kang Feng

Department of Computing and Information Systems, University of Paisley, High Street, Paisley, PA1 2BE, United Kingdom

Abstract

Data handling is an indispensable issue in information systems (IS) development. Traditional data handling assumes an objectivist philosophical position, which may be responsible for some problems in this area. This paper looks at this issue from a perspective of sign, information and meaning, and presents a framework for data handling based on a subjectivist position.

1. OBJECTIVIST DATA HANDLING

A number of different and contrasting perspectives have been adopted in IS development (Avgerou and Cornford 1993). In the mainstream of structured methodologies, one perspective is concerned with modelling the world in terms of the items about which data is stored - referred to as the *data modelling* approach. Like other traditional approaches, data modelling has said little explicitly about its underlying philosophy, but in practice it assumes an objectivist position - data and information are independent of their producers neutrally reflecting real world structures (Lewis 1993; Mingers 1992; Date 1990, Page 600; Tsichritzis and Lochovsky 1982, Page 3).

This objectivist position seems responsible for the lack of adequate insight of the nature of data and the lack of systematic methods for handling them. Different stake-holder's viewpoints regarding data are usually not well respected or preserved during all stages of data handling. Data modelling languages and methods do not provide mechanisms for the explicit capture of them either. Database management systems usually do not provide adequate facilities for the representation and derivation of different interpretations of data. In addition, the development of data models appear to have been heavily influenced by the concepts of data structures for computers, such as *record, field* and *tree structure*, the most widely used of them can be regarded as a direct outgrowth of the data representation technology for computers. Thus the whole situation still seems immature as Bubenko and Olive observed in 1986 (Bubenko and Olive 1986). Our understanding of data, information and information systems is rather shallow and patchy, the concepts used to tackle the problems involved in this area are too general and vague, the practice of data modelling remains, to a large extent, as a 'black art' (Flavin 1981), rather than a science.

2. RELEVANT THEORIES

The data modelling approach to IS development is related to a 'bigger' issue, namely *data handling*. It seems useful to formulate our thoughts and methods for handling data in terms of the following items: some *fundamental views on data*, and some *data handling methods* which in turn consist of *modelling languages* and *modelling procedures*. In the case of handling data by means of computers, we may add some *implementation mechanisms*.

It is vital to develop a sensible fundamental view on data. It would be ideal if data handling methods could be based on a science of data, one like natural sciences which would discover and embody laws or trends that govern the regularity of data. Unfortunately, data are not an objective physical property of the natural world, rather, they are products of human activities. For this very reason, a restricted science of data does not exist, nor it ever will. Thus we have to find alternative ways of handling data. Here a 'Three Worlds' theory seems rather enlightening and useful (Popper 1972, Thompson 1989; Mingers 1992).

Popper (Popper 1972, Page 74) proposed to split the world into three realms: World 1 is the world of real things, of things in and of themselves, and they exist independently of any particular observer. World 2 is the subjective world of the individual. It represents the world of our conscious experiences, and it is peculiar to that individual. World 3 contains all systems that have some ordering logic in terms of which the system may be understood. As such, it contains logic and all systems that aim to be logical. It contains science; it also contains the arts.

Thompson (Thompson 1989) further pointed out that World 1 and 2 may interact, and World 2 and 3 may interact; however, no mechanism exists that allows direct interactions between World 1 and World 3. World 2 is fragmented; each individual exists in isolation, having no direct access to the World 2 of some other individual. People can communicate essentially only through World 3.

Mingers (Mingers 1992) examined theories of information and meaning, and presented a 'Three Worlds' view. There are three different 'worlds' to which any utterance is related - the objective world of external nature which obtains or can be brought about; the subjective world of internal nature, constituted by people's private experiences; and the social world of society consisting of consensual, intersubjective practices and norms. *The* world, *my* world, and *our* world.

Mingers utilises a number of foundation theories including Maturana and Varela's cognitive theories, Habermas' work on critical theory and a communicative theory of social action and Dretake's account of information, and develops a theory of information and meaning (Mingers 1992). This theory can be summarised as follows:

Any event in the world carries information - information about its own origins. The amount of information reflects the reduction in possibilities brought about by the event. This information is independent of any observer. Information, whether carried or not by something other than the event itself, is an objective feature of the world in the same way as are physical objects and their properties.

Information may be transmitted. Signs are the most primitive elements which convey information. A sign can carry information about something other than itself. A direct physical sign only carries information about that which caused it. However a linguistic sign carries information both about what it describes - its *propositional content* - and about its production by a particular speaker - its *illocutionary content*, or what it is supposed to achieve.

Information is objective, whereas meaning is either subjective or intersubjective, the latter is derived from the former. The physical world, in which all signs and signals are ultimately embodied, is essentially analogue - a continuum of differences rich in information. The transformation of information into meaning involves a digitalisation of the analogue. It is the knowledge, intentions, context of the receiver that is determining what particular aspects of the available analogue information are being digitalised into meaning. Meaning is the *semantic content* of an information source.

3. OUTLINE OF A SUBJECTIVIST APPROACH TO DATA HANDLING

It may shed some light on data, data modelling and databases if they are looked at from a broader perspective of sign, information and meaning and the 'Three World' theory.

Information

It appears that any distinct thing including signs has a triggering force. When applied on the sensory organism of a live object, it becomes a stimulus for *connotation*; in the case of humans, the stimulus also indirectly triggers *intention* via *connotation*. This is a process of cognition. It may or may not create stimulus, depending on the sensory organism, but the force exists regardless anyway, so it is objective.

This process of cognition can be regarded as a complex transformation process, the material that is being gathered, transferred, and transformed during this process may be what we call *information*. This is the view of the information-processing approach of cognitive psychology (Best 1992, Page 22).

Information may be carried by either events or things themselves or signs. Data is a type of linguistic signs. Data handling therefore serves as a basis for information processing and meaning appreciation. In order to understand information systems, we must understand data and ways of handling them.

Data Is A Type Of Sign That Conveys Information

Data is not a substantial physical property of the real world that is independent of any observer including whoever creates the data, so data is not of World 1. Rather, data are recordings of the real things or humans' thoughts. Following Stamper (Stamper 1973, 1985, 1987) and Mingers (Mingers 1992), we take signs as the most primitive elements which convey information, and data can be regarded as a particular type of linguistic signs.

Data may be recordings that belong to some individual, so they represent World 1 experienced by the individual and only make sense to the individual who creates them. This kind of data is of World 2 - the subjective world. Data may also be recordings of things and are organised by means of some concepts and rules which are acceptable by a community of people who have similar experiences, expectations and knowledge. This kind of data can therefore be understood by the community - they trigger some basic common intersubjective meaning in their minds. Data that are organised this way are objects of World 3 - the intersubjective world. This common meaning then triggers connotations and intentions whereby individual World 2 subjective meanings of data are arrived at. This is the process of data interpretation.

Data is a product of human activities, it is part of human's social reality. The social reality is a mixture of objective and subjective phenomena, ever-changing and complex, so is data. In general, to tackle complexity, humans create models of the reality, physical models, quantitative models or conceptual models, to reduce the multiformality of the world to some common forms so that things can be brought into a logical and conceptual relationship with each other. Then they use the models to test hypotheses and reveal laws or trends which govern the regularities and continuities of the real world whereby to understand the complex reality. These models serve as *intellectual constructs* pertaining to the complex reality. For handling data, our goal is also to find common forms of data, so that data created in deferent situations can become compatible. It may achieve sensible understanding and interpretations of data if some intellectual constructs of data and some procedures of pertaining them to complex reality from which data result can be developed. The former is in general called *data models*, and the latter *data modelling*. They are concepts, rules and procedures for the identification, structuring and manipulation of data. A common practice in this area is to model the objects in a domain of interest, which either perform actions or receive actions or both.

A data model can be developed, understood and used by a particular individual. In that case, the data model is a tool for the handling of world 2 data. As far as ISs go, what is worth being looked at are those data models that are capable of being accepted by a community of people. In this case, a data model is a mechanism for the generation of a World 3 intersubjective system, which should in turn provide the individuals within the community with a common base for the derivation of required World 2 subjective meaning.

Database systems are a type of computer implementations of the systems of data. If and only if it provides its targeted users with a basis for them to derive required meaning, a database system can be regarded as satisfying the information requirements of the users. This gives rise to the need of data modelling which emphases subjective interpretation in data handling. That is, how to construct a common intersubjective system of data which provides individual users with a context and mechanism for the derivation of their own subjective meanings of data. To this end, a number of problems must be resolved. The discussion in the following sections are based on the author's previous work (Feng and Rooms 1991, Feng 1993a, 1993b, and 1993c).

The Capture Of Different Stake-holders' Raw Data Needs

The common base of information is an integration of different stake-holder's data needs in terms of some data model accepted by the community of the stake-holders. A systematic method is required to elicit and identify these needs. The soft systems methodology (SSM) can be used here. SSM is a methodology based on the interpretivist thinking. It uses systemic *holons* - the 'relevant human activity systems' (Checkland 1981) to make explicit and formalise stake-holder's different views on a complex domain. These holons may be of 'primary task', in that case they are closely related to the ontology of the domain. These holons may also be 'issue based' under the same or different *waltenschauung*. They reflect directly user's perceptions, views and requirements toward the domain in question

These holons are formally defined by *root definitions* which usually consist of CATWOE - six elements. Then a set of sufficient and necessary activities are identified whose executions will materialise the root definition of a holon. This is called a *conceptual model* of the relevant system. The stake-holders and analysts will identify the sufficient and necessary conditions for the execution of an activity involved in the conceptual model. A set of inference rules may be developed to analyse the logical relationships between the activities, and a *minimal cover* - the simplest structure of a conceptual model can be arrived at.

An activity in a conceptual model is also a holon. So a set of sufficient and necessary activities whose executions will materialise this holon can be identified, analysed and structured in the same way the top level holon is processed. This process is used recursively, so the conceptual model will be further decomposed till the lowest level is arrived at, namely a holon composed of *elementary activities*.

An elementary activity is atomic and cannot be decomposed further. It is defined by means of the following conditions:

- A unique unit of work,
- Consisting of a set of serially performed steps,
- All steps directed towards a common goal,
- All steps utilising or creating a common set of data.

An elementary activity is essentially the simplest process of transformation - To change the status of some properties of a set of specified objects for some reason. The identification of an elementary activity relies heavily on the analysis of the data involved. This data analysis is conducted at both *type* level and *instance* level. At the type level, a certain set of data types will be required for determining the types of the *object and their properties* involved in a task, so they remain unchanged through out the activity. At the instance level, the data for determining an object instance should be the same immediately before and after the activity, whereas the data that describe the status of some properties of the object will change, but only once for one elementary activity.

The elementary activities must be defined as being mutually exclusive. They should not overlap, nor are they allowed to be nested. When identifying data involved in an elementary activity, only sufficient and necessary data for the transformation qualify. These data are of only two kinds: either those which are used for determining the objects to work on, or those which are used for achieving a particular goal other than determining the objects.

This way, for each of stake-holder's holons, a set of sufficient and necessary data for all elementary activities within the holon are found. They are not interpreted and integrated into a common data system yet, they are of world 2. We call them stake-holder's subjective *raw data needs*.

A Subjectivist System Of Concepts And Rules For Intersubjective Data Systems

In order to construct a common system of data from many sets of raw data needs, a system of concepts and rules that are acceptable by the community of the users of the information system in question must be developed and applied to the raw data needs. These concepts and rules will be used to formalise user' subjective and intersubjective worlds, so their underlying philosophy should be subjectivist rather than objectivist. The concepts and rules should be general enough in order to impose little constraints on the stake-holders in deriving their subjective meanings of data. They must also have unambiguous definitions so as to provide a sound basis for the analysis and integration of the raw data needs. Such a system of concepts and rules are possible only if based on a commonly acceptable *fundamental view* on domains of interest.

Our solution is, for the purpose of data handling, to view *a domain of interest as composed of objects*, and we use only two concepts as the building blocks of our intersubjective data system, namely *object* and *information of interest* about object. The following are their definitions:

Object: - A thing is defined as an *object* if and only if

a) it is distinctly identifiable, AND

b) we want to talk about it, AND

c) EITHER it has information of interest that are to be captured, OR it is referred to by at least one different object, OR it is related to at least one different object, OR any combinations of them.

Information of interest: - A piece of information is defined as *information of interest* about a defined object if and only if

a) it is concerned with the object, AND

b) we are interested in it, AND

c) it does not satisfy the conditions for an object.

A piece of *information of interest* may be concerned with some property of the object, in that case, the information *describes* the object. On the other hand, a piece of information may lead to some other object, which then captures some *relationship* between objects. A piece of information of interest will have a type and values. For the latter case, the values will be drawn from the instances of the other object. This serves as a mechanism for the integration of objects into a whole and for the retrieval of information regarding objects from different point of view via different *perceptive objects*.

Notice that these two concepts are profoundly different from those conventional concepts, namely *entity, property* and *relationship* in terms of their underlying philosophy. The latter are objectivist, they are supposed to be used to neutrally describe the real world structure, though this will never be fully achieved. The former are subjectivist, they unequivocally acknowledge that what are to be captured are what the user 'wants to talk about' and what 'interest's the user. Compared with conventional data modelling concepts, fewer modelling constructs in our system also narrows the scope of design possibilities. Furthermore, the propositional content of these two concepts is simple, unambiguous and clearly different, so the *semantic relativeness* can be eased. These two concepts are further classified according to the basic methods of human organisation (Coad and Yordon 1990, Page 16), and as a result, a powerful modelling mechanism can be established (Feng 1993c).

Data Statistical Characteristics

Now we have subjective raw data needs identified, and concepts and rules for intersubjective data systems defined, but a gap between these two exists. A method is required here to interpret raw data in terms of those concepts. Our solution is to find what we call data statistical characteristics (Feng 1993b).

While subjective raw data needs are being identified, the elementary activities which use the raw data are identified along with them. These two therefore form a structure which we call *data and elementary activity pairs*. The relationship between these two and the relationship between raw data against the background of elementary activities can be statistically analysed. A set of parameters are conceived to formally describe these relationships and serve as a mechanism for the statistical analysis. They are defined as follows.

- Parameter TU (usage across elementary activities): The number of elementary activities by which a data type is used.
- Parameter JU (joint usage): The number of other data types with which a data type is used.
- Parameter UR (usage ratio): The ratio between the number of other data types with which a datum is used in most tasks and the total number of other data types with which the data type is used.
- Parameter JUR (joint usage ratio): Data type A's JUR with data type group B is defined as - (The number of the tasks where datum A and data group B are used together) / (The number of tasks where data A is used).

The relationship between JUR and UR is: For any datum, say, Data type Di

URi = Total Number of JURhigh_ix / JU, where URi is the UR of Data type Di, and JURhigh_ix means a high JUR of Data type Di with another Data type Dx.

We found that the probability distributions of TU, JU and UR have the following curves.

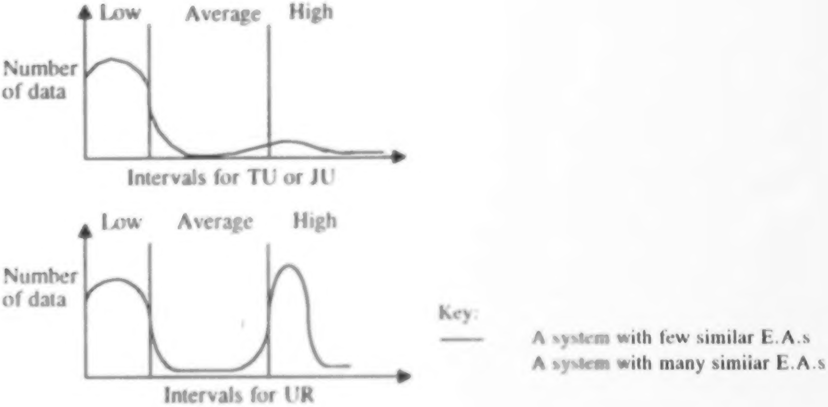


Figure 1. Probability distributions of TU, JU and UR

With obvious turning points which have high differential values, these curves can be divided into three regions: high, average, and low. These regions form a set of criteria for the classification of raw data types into *primary data* or *auxiliary data*. They are defined as follows:

Primary data

A datum used in determining a particular type of objects involved in an elementary activity. It is a determining part of the representation of an object.

Auxiliary data

A datum used as an auxiliary material in achieving the particular goal (i.e., some change of the status of some object(s)) of an elementary activity. It is an non-determining part of the representation of an object.

A domain of interest has a basic structure, which can be regarded as composed of a number of objects. An object is likely to be involved in many elementary activities, involved with many other objects, and independent. A primary datum is a determining part of the representation of an object, so it should have a high TU, a high JU, and a low UR. Otherwise a raw data type is classified as an auxiliary datum. An auxiliary datum, depending on the nature of the tasks, may have a different parameter in terms of being high, average, or low. Where there are few similar tasks, an auxiliary datum will have a low TU, a low to average JU and a high UR, however, where there are many similar tasks, both its TU and JU may be average though its UR may still be high. Therefore the curves for the first three parameters may be slightly different. Figure 2 illustrates this situation. With the criteria and parameters of each datum at hand, to classify them is trivial.

Primary data lead to the identification of *objects*, and auxiliary data to the *information of interest* of objects. The fourth parameter JUR is used to find the attachment of a piece of information of interest to an object or relationships between objects.

In summary, the statistical characteristics of data in terms of the probability distributions of the parameters provide a sound basis for some interpretation of the *raw data needs* of the stake-holders. On the contrary, with conventional data modelling and object-oriented analysis, the identification of entities or objects seem rather arbitrary. This may reflect an inherent contradiction within those methods: they assume an objectivist position, but at some important stage the user of these methods have to solely rely upon subjective observations.

The Integration Of Local Data Schemas

Once different stake-holder's raw data needs are processed by means of the above procedure, they are transformed into a compatible form and become intersubjective. But they are still *local data schemas*. These local schemas should be integrated into a *global one*, which serves as the definition of a common information repository. Having said that, it is worth noting that the purpose of schema interpretation is purely of implementation consideration, rather than conceptually imposing a global frozen interpretation of data upon all users. On the contrary, different user's interpretation of data should be preserved, and moreover, the structure of the data system should be flexible enough to cater for the inevitable changes of user interpretations of data in terms of their 'relevant systems'. Due to the simple modelling concepts of our method, the integration process is merely concerned with object integration, which makes use of the concepts of *identity* and *generalisation*.

Identical objects are those which have identical information of interest, whether they have identical names or not, where 'identical information of interest' refer to those which have same semantics and same sets of values. If an object from local data schema 1 is found identical to an object from local data schema 2, neither of these objects are allowed to participate further in any other identity integration between these two local data schemas. The reason is that each object is unique within its local data schema.

Similar objects will be integrated into a super/subclass hierarchy. There may be three different situations:

First, all 'information of interest' of one object is a subset of those of another object. The former should be defined as a super class of the latter.

Second, the intersection of the 'information of interest' of two objects is a subset of those of both objects. This may give rise to a super class object which takes the intersection as its 'information of interest', and the two objects are its sub-classes.

Third, two objects have no common 'information of interest' but have similar semantics. It may be appropriate to define a super class for these two objects.

Note that a local schema is integrated into the global one, it does not disappear, because all its objects are preserved. Therefore, user' subjective interpretations of data are intact.

A Mechanism For The Ready Derivation Of Subjective Meaning

Through the integration process, data in the global data schema is organised in hierarchical structures. These hierarchical structures are inter-related by means of the 'information of interest' of the 'relationship' type (see page 6). That is to say, a piece of information of interest, say I_1 of object A, takes as its values the instances of object B. I_1 captures a relationship of A with B. B will have a piece of information, say I_2, which takes as its values the instances of A, and it is a reverse of I_1.

Not only is this mechanism used to integrate different hierarchical structures, but also it serves as a means for individual stake-holders to readily derive subjective meaning of data as they perceived in their 'relevant systems' in the first place. They can interpret the common data system from the viewpoints of the objects in their world 2, and take the relevant part of the rest of the data system as direct or indirect values of the information of interest of these objects. These objects may be termed as perspective objects of the stake-holders.

Notice that our common data system is not a neutral account of the structure of a domain of interest independent of observers, it is never meant to be. On the contrary, it is distilled from an aggregation of subjective observations and is a sign system. The information conveyed by this sign system will trigger subjective and intersubjective meaning. Therefor we claim that the underlying philosophy of our framework of data handling is not objectivist, but subjectivist.

4. SUMMARY

Some problems with contemporary data handling approaches might stem from their underlying objectivist philosophical position. Further progress may require the paradigm shift from objectivist to subjectivist. This paper looks at the issue of data handling from a perspective of sign, information and meaning, and presents an outline of a subjectivist approach. The main body of the approach was developed through case studies, detailed accounts of which can be found in a number of publications (Feng and Rooms 1991, Feng 1993a, 1993b, and 1993c).

5. REFERENCES

Avgeron, C., and Cornford, T., (1993). A Review of the methodologies movement. *Journal of Information Technology* (1993) 5, pp. 277-286.

Best, J. B., (1992). "Cognitive Psychology", 3rd Edition, West Publishing Company, St. Paul.

Bubenko, J.A., and Olive, A., (1986). Dynamic or temporal modelling- an illustrative comparison. *Syslab working paper* 117, Syslab, Univ. of Stockholm, Stockholm.

Checkland, P., (1981). "Systems Thinking, Systems Practice", John Willy & Sons, Chichester.

Coad, P., and Yourdon, E., (1990). "Object-Oriented Analysis", Yourdon Press, Prentice-hall. Englewood Cliffs.

Date, C.J., (1990). "An Introduction to Database Systems", Fifth edition. Addison-Wesley Publishing Company, Reading.

Feng, J.-K., and Rooms, H.W., (1991). A systems based modelling technique for KBS construction, *in* "Systems Thinking in Europe", Jackson, M.C. et al ed., Plenum, London.

Feng, J.-K., (1993a). Conceptual modelling and DBS/KAB design, *in* "Systems Science: Addressing Global Issues", Stowell, F. West, D. and Howell, J. ed., Plenum, London.

Feng, J.-K., (1993b). Function-oriented data modelling, *in* "Systems Science: Addressing Global Issues", Stowell, F. West, D. and Howell, J. ed., Plenum, London.

Feng, J.-K., (1993c). "Conceptual Modelling for the Construction of Database and Knowledge-Based Systems", MPhil thesis, University of Portsmouth, United Kingdom.

Flavin, M., (1981). "Fundamental Concepts of Information Modelling", Yourdon Press, A Prentice-Hall Company, Englewood Cliffs, NJ 07632.

Lewis, P. J., (1993). Linking soft systems methodology with data-focused information systems developments, *Journal of Information Systems* 1993 (3), pp. 169-186.

Mingers, J., (1992). An examination of information and meaning, *2nd UKSS Information Systems Semina*.

Popper, K.R., (1972). "Objective Knowledge", Oxford Press, Oxford.

Stamper, R., (1973). "Information in Business and Administrative Systems", Wiley, London.

Stamper, R., (1985). Towards a theory of information. Information: mystical fluid or subject for scientific enquiry? *The Computer Journal* 28, 3, pp. 195-199.

Stamper, R., (1987). Semantics. *in* "Critical Issues in Information Systems Research", Boland, R. and Hirscheim, R. ed., Wiley, London.

Thompson, J.P., (1989). "Data with Semantics", Van Nostrand Reinhold, New York.

Tsichritzis, D. C., and Lochovaky, F. H., (1982). "Data Models", Prentice-Hall Inc., Englewood Cliffs.

A Revised Spiral Model for Object-Oriented Development.

Egbert van der Walt[1] and Annette L Steenkamp[2]

1 Technology Department, S A Post Office. PO Box 10000, Pretoria, 0001, South Africa
2 Department of Computer Science and Information Systems, University of South Africa, PO Box 392, Pretoria 1000, South Africa

Abstract

A life cycle model for object-oriented development is proposed The Spiral model as proposed by [Boehm 86] is updated to make provision for object-oriented development The updated spiral model comprises five cycles namely the feasibility, architecture, analysis, design and implementation cycles The four quadrants of each cycle of the spiral are also described

Keywords

Software development life cycle, object-oriented development, software process model

1 Introduction

Various authors have published work on object-oriented development (OOD) [Booch91, Rumbauch91, Meyer89] Although the OOD methodology has been described by these authors, the effect of OOD on the total software life cycle has not received much attention Some of the important properties of OOD identified by Booch and Rumbauch are

- OOD is a evolutionary process
- It is an iterative rather than sequential process
- It is a seamless process where the separation of life cycle phases is not distinct
- Much of the development effort is put into analysis

The waterfall life cycle proposed by Royce [Royce70] or variants of the model proposed by various authors [Boehm76, Yourdon83, Grahm89] do not make provision for all the identified properties It is clear that a need exists for a life cycle framework which describes and supports the activities and properties of OOD

Because the separation of life cycle phases in OOD is not distinct, a life cycle model with a risk management and strong control components is needed

The spiral model proposed by Boehm [Boehm86] makes provision for evolutionary development with risk analysis and a control component Although the Spiral Model was not specifically intended for OOD, it was adopted as a basic model for OOD due to the features described in this paper This model embodies many of the features of other life cycle models and adds risk-driven and prototyping strategies

In this article the modification of the spiral model is proposed to adapt it to OOD by additional emphasis on analysis, inclusion of activities associated with OOD and adding more control checkpoints to the model This article follows as a continuation of work published by Du Plessis and van der Walt [Du Plessis van der Walt 92]

In section 2 the generic life cycle model which defines the generic actions taken in each of the quadrants, is described The five cycles of the revised spiral model for OOD is explained in section 2

2 The Generic Life Cycle Model

The generic model of a system development life cycle phase as viewed in Figure 1 has four quadrants Generic tasks to be undertaken during each of the quadrants have been identified by following Sage and Palmer's systems approach [Sage and Palmer 90]

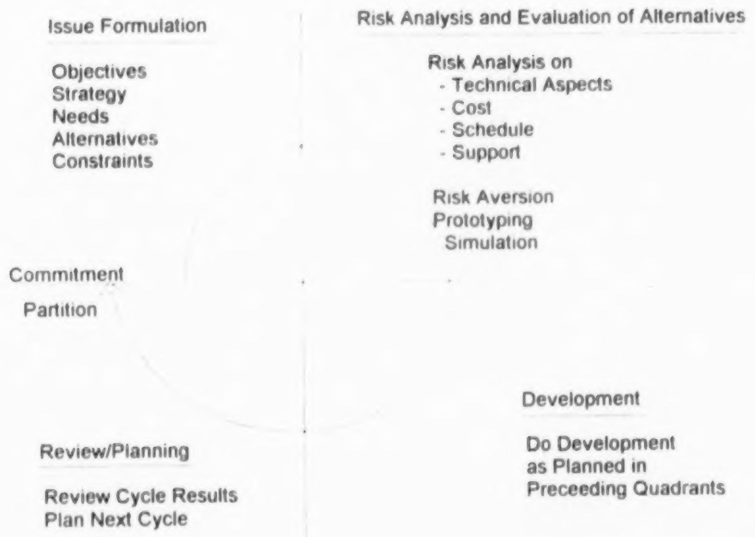

Issue Formulation
Objectives
Strategy
Needs
Alternatives
Constraints
Risk Analysis and Evaluation of Alternatives
Risk Analysis on
- Technical Aspects
- Cost
- Schedule
- Support
Risk Aversion
Prototyping
Simulation
Commitment
Partition
Development
Do Development
as Planned in
Preceeding Quadrants
Review/Planning
Review Cycle Results
Plan Next Cycle

Figure 1. Quadrants of the Spiral Model

Quadrant 1 - Issue Formulation, during which the following tasks are performed

a) Identify the objectives of the cycle

b) Formulate a strategy to reach the objectives according to plan

c) Identify alternative strategies to reach the objectives

d) Identify applicable constraints

Work in this quadrant culminates in the strategy definition review where the objectives and strategies are reviewed

Quadrant 2 - Risk Analysis and Evaluation of Alternatives [Hofkin and Powell 91] identified three sub-phases

a) Risk analysis Evaluate the identified strategies and determine the risks involved for the technical aspects, costs, schedule and support This phase culminates in the risk review where the strategy with the lowest risk is chosen The lowest risk is defined according to the priorities set on the evaluated risk areas

b) Risk aversion planning Planning is done to lower the identified risks Alternative handling of the high risk areas may be identified

c) Prototyping Various modelling techniques such as prototyping, simulation and analytic modelling may be used to test areas of risk such as the estimated communications load and database performance Prototyping helps to reduce the areas of risk and enhance the project participant's understanding of the system

The risk analysis phase ends when a commitment to a specific strategy is made

Quadrant 3 - Development, during which the development activities and tasks of a cycle are performed Provision is made for evolutionary development as the development quadrant can be visited for each module of code or subsystem under development

Quadrant 4 - Review Planning This quadrant may be divided into two main sub-phases

a) Evaluation, during which the deliverables produced during the third quadrant are verified against the specifications and a decision is made whether to update the project baseline Quality related activities like code walkthroughs are performed The evaluation phase ends when the product development review is done The result of this review is the acceptance of the developed product

b) Planning for the next cycle of development is done A review of the work breakdown structure (WBS) and cost breakdown structure (CBS) is done Software cost estimations are updated The schedule for the next phase is drawn up This quadrant ends when a software development plan for the next cycle is presented to the customer and the commitment to continue with the next cycle is obtained from the user

3 The Revised Spiral Model for OO Development

The revised spiral model was obtained by changing the spiral model proposed by Boehm [Boehm86] to make provision for the processes of OO development. The model consists of five cycles in a spiral. The project starts with the first cycle from the centre of the spiral. As development progresses, consecutive cycles are completed until the final acceptance and completion of the project.

Work in each quadrant of the cycle follows the generic format of the spiral model outlined earlier, namely Issue Formulation, Analysis and Evaluation of Alternatives, Development and Review/Planning. Figure 2 depicts the revised spiral model.

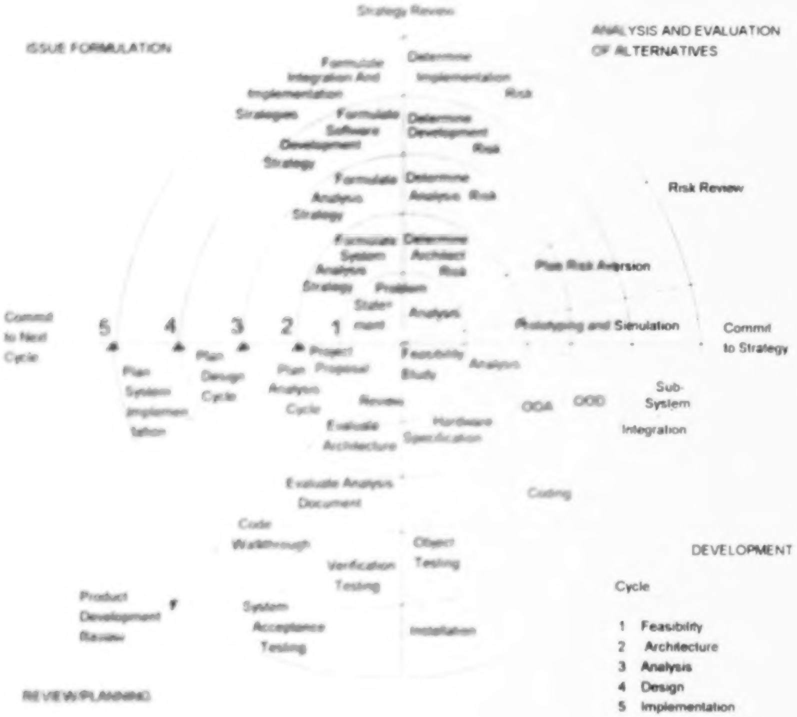

Strategy Review
ISSUE FORMULATION
ANALYSIS AND EVALUATION OF ALTERNATIVES
Formulate Integration And Implementation Strategies
Determine Implementation Risk
Formulate Software Development Strategy
Determine Development Risk
Formulate Analysis Strategy
Determine Analysis Risk
Formulate System Analysis Strategy
Determine Architect Risk
Problem Statement
Analysis
Risk Review
Plan Risk Aversion
Prototyping and Simulation
Commit to Next Cycle
5
4
3
2
1
Commit to Strategy
Plan System Implementation
Plan Design Cycle
Plan Analysis Cycle
Project Proposal
Feasibility Study
Analysis
Review
Evaluate Architecture
Hardware Specification
OOA
OOD
Sub-System Integration
Evaluate Analysis Document
Coding
Code Walkthrough
Object Testing
Verification Testing
DEVELOPMENT
Product Development Review
System Acceptance Testing
Installation
REVIEW/PLANNING
Cycle
1 Feasibility
2 Architecture
3 Analysis
4 Design
5 Implementation

Figure 2. Revised Spiral Model for OO Development

The model consist of the following cycles

3.1 Cycle 1 - Feasibility

The cycle commences with a stimulus representing the need for a software system, or an enhancement of an existing system During Issue Formulation a Problem Statement is formulated, which includes a high level description of the environment where the system is to be implemented, as well as constraints which are relevant to the project During problem analysis and evaluation, high risk areas are identified and alternative approaches to address the requirements evaluated A Feasibility study is conducted in the Development Quadrant involving the first cost benefit analysis, technical feasibility evaluation and investigation of the legal implications of the project The Feasibility Study culminates in a Preliminary Project Proposal The Preliminary Project Proposal contains a detailed problem definition within the context of the business, a high level software system strategy which describes the planned approach which will be followed to address the problem definition, an overall estimate of whether the identified user needs can be satisfied using currently available software and hardware resources, an analysis of the cost-effectiveness of the proposed system, and whether or not it might be developed given existing time and budgetary constraints The Feasibility Cycle may be traversed more than once if the proposal is not accepted and changes to the high level requirements are made Changes to the requirements may have an effect on the approach taken to the development as well as the development costs Once the Preliminary Project Proposal is accepted and authorized by management the project enters the Architecture Cycle

3.2 Cycle 2 - Architecture

In most of the cases, the Feasibility Cycle is done in a short time, which often does not allow for thorough analysis Considering that project proposals are most of the time composed by marketing people who are not always aware of technical pitfalls, it may be very risky to award a multimillion dollar contract for full system development, based on the Preliminary Project Proposal alone It is therefore recommended that a contract for the next cycle, ie the Architecture Cycle only be awarded after the Feasibility Cycle This approach is of benefit to both the developer and the customer

In the Architecture Cycle the top level system software architecture and hardware architecture are determined Once this has been done, the estimates may be reviewed and a tender for the completion of the system considered This will drastically lower the risks of the system as the hardware configuration will be known The variance on the estimated hardware cost will be minimal

When the software architecture is known, most technical pitfalls will be known and the risks of under estimation should be less

The Architecture Cycle starts with the formulation of objectives for the cycle This cycle must produce the hardware architecture as well as a top level software model Starting with information obtained from the Preliminary Project Proposal, various system architecture strategies are identified Hardware architecture issues such as the type of network, hardware platform, hardware support etc are identified The software system strategy defined in the preliminary proposal is elaborated upon The first quadrant of the cycle is completed when the strategy definition review is held where all the strategies are reviewed

Risk analysis is performed on the strategies The risk areas of the strategies are identified During the risk review, the strategy with the smallest risk is selected for further evaluation Risk aversion planning is done or the strategy may even be changed to a limited extent in order to reduce risk Prototypes or simulations of critical areas and areas identified in the risk analysis of the system may be done to reduce risk and to enhance the understanding of the proposed system Bench marking for hardware evaluation may also be done The second quadrant ends when a commitment to a specific strategy is made

The third quadrant involves top level Analysis [Booch 91], [Coad and Yourdon 90], [Rumbaugh et Al 91] during which the high level user requirements, first informally documented in the Preliminary Project Proposal, and the acceptance criteria are specified more formally in the architecture document This document forms the basis of the contract between the client and the software developer Top level class diagrams, data flow diagrams and state diagrams are produced The top level class diagrams forms the basis of the object model, data flow diagrams, the basis of the functional model and the state diagrams the basis of the dynamic model as defined by [Rumbauch et Al 91] At this stage the sub-systems may be identified which models the top level object, functional and dynamic perspectives respectively Sub-systems may be seen as very high level objects with low coupling to other high level objects and with a specific function The hardware configuration is also defined This involves the definition of required hardware like the CPU, printers, disk drives, LAN interfaces etc as well as the network design and allocation of peripherals to workstations The Architecture document produced in this cycle consists of the class diagrams, data flow diagrams, state diagrams, sub-system specification, and hardware definition specification

In the fourth quadrant, the architecture is evaluated in order to make sure that all issues are covered and that the hardware architecture ties up with the software architecture The architectural design is completed when the architecture has been accepted at the product development review of the system architecture

The Architecture Cycle concludes with the planning of the analysis phase as well as the rest of system development The WBS, CBS and schedules are updated and a project proposal for the development of the next cycle as well as the rest of the system is compiled The project management plan for the rest of the project is defined The Architecture Cycle ends when the project proposal is accepted by the client

If it is realized during the analysis or design cycles that the hardware specification will not meet the system demands, this cycle may be revisited

3.3 Cycle 3 - Analysis

The issue formulation quadrant for the analysis cycle is scaled down in comparison with the Architecture Cycle The main guidelines for the analysis strategy is defined in the architecture cycle The top level class diagrams already exist which leave the analysis cycle with the task to complete the OOA using the diagrams defined in the architecture cycle Objectives, strategies and constraints for the analysis of the sub-systems are set The first quadrant is completed with the strategy definition review

As in the Architecture Cycle, risk analysis is done for the defined strategies The sub-system strategies with the lowest risk are identified during the risk review Risk aversion planning is done for the identified high risk areas

Prototyping of the top level class structure may be done to ensure the feasibility of the strategy Identified risk areas may also be prototyped to resolve uncertainties The risk quadrant ends with the commitment to a set of strategies for the sub-systems

The next quadrant of the cycle involves more detailed OOA The top level class diagrams , data flow diagrams and state diagrams defined in the Architecture Document are used as the starting point in the definition of the object model, dynamic model and functional model The result of the analysis is the three models which describe the system

In the fourth quadrant the three models are evaluated to ensure that they are consistent and complete OOA is completed once the analysis document is approved at the product development review

The Design Cycle is planned and the WBS and CBS are updated Estimation figures are updated for the rest of the system development The software development plan is updated The Analysis Cycle ends when customer agrees to the continuation of the design cycle

3.4 Cycle 4 - Design

During this cycle the system is designed and sub-systems developed In the first quadrant the objectives for the Design Cycle are defined These objectives may be influenced by the system requirements For example the design objectives of a batch processing system that requires maximum security to data will be different from a real time system where the emphasis is on the processing speed The design strategy is formulated when the order of design of sub-systems and parts within the sub-systems are defined This order is influenced by the integration / test plan The classes which form the kernel of the system must be designed and developed first as they will be used in subsequent development In order to spread the system testing activity throughout the design and implementation cycle, testable modules must be produced by the software development team The first quadrant culminates in the software development strategy review

Risk analysis is done and high risk practices are identified Risk aversion plans are made and changes are made to the strategy where needed Prototyping of high risk aspects are done to reduce uncertainty and to enhance the understanding of the system The second quadrant ends when the design strategy is accepted The structure of the basic architecture for the system and the solution strategy chosen are documented in the System Design Document

OO development is an incremental process This implies that the steps in development quadrant may be re-iterated before a subsystem is completed The detailed physical design is documented in the Design Document Steps of OO design, as defined by [Rumbaugh et Al 91] are

a) Combine the three models obtained in the Analysis Cycle to implement operations on classes
b) Design algorithms to implement operations
c) Optimize access paths to data
d) Implement control for external interactions
e) Adjust class structure to increase inheritance
f) Design associations
g) Determine object representation
h) Package classes and associated modules

Incremental prototyping may be used as part of the iterative and evolutionary strategy of OOD, referred to by [Booch 91] as "Round-trip gestalt design"

Throughout the Design Cycle the behavior of the prototypes that are built are evaluated to determine if the user requirements are met, and whether these prototypes may be integrated into the evolving product. Progress on the development of the classes of the logical system is tracked by middle management by means of formal and informal design reviews with the development team. Progress is measured by logging the classes in the logical design and the modules in the physical design which have been completed and are functioning correctly. At the same time the stability of key interfaces may be used to measure the progress towards the final product.

Once a subsystem is completed verification testing is done on the subsystem as a whole. The code is accepted after the product development review. Need might arise during this cycle to re-iterate the Analysis Cycle, resulting in a revision of the development strategy.

As sub-systems are completed, planning for sub-system integration and system implementation is done. The software management plan is updated. This implies that the WBS, CBS and cost estimations are reviewed. The Development Cycle ends when the customer agrees to proceed with implementation. Each identified sub system goes through its own scaled down development spiral, consisting of the analysis and development cycles.

3.5 Cycle 5 - Implementation

During Issue Formulation, various strategies for subsystem integration and integration testing are identified. Different ways of installing the hardware of the final system are determined. The training programme of the new system users is also drawn up. During the analysis and evaluation of the alternatives, the risks involved in the different subsystem integration strategies are identified. Integration tests are evaluated to ensure that they are adequate. The hardware installation layout and software packaging are reviewed. The contents of the training programme is reviewed to ensure that it covers all aspects of the system. During the development quadrant subsystem integration is performed and tested. Subsystem integration may reveal incompatibilities between subsystem interfaces requiring the Design Cycle to be repeated. Progress made with integration is continuously monitored. Finally the system acceptance testing is done Once the customer is satisfied that the system runs according to specifications, the system is considered completed

4 Life cycle summary

A variation of the Spiral Life Cycle Model is proposed for OO development This model consists of five cycles the Feasibility Cycle, Architecture Cycle, Analysis Cycle, Design Cycle and implementation cycle Strong emphasis is laid on risk analysis before actual development work is done This approach reduces the chance of project failure Various reviews are held throughout the development life cycle as checkpoints to ensure control on the development process The development process may move to a previous cycle when needed A Preliminary Project Proposal is prepared during the Feasibility Cycle after which a contract for the Architecture Cycle is awarded During the Architecture Cycle, the foundation is laid for the object model, dynamic model and functional model The hardware specification for the system is defined The full contract for system development is awarded only after the hardware platform and software architecture has been determined OOA is done during the Analysis Cycle This cycle results in the object model, dynamic model and functional models During the Design Cycle, sub-systems are designed and developed The result of this cycle is the detailed object, dynamic and functional models as well as the coded sub-systems During the implementation cycle, integration of the top level sub-systems are done and the system are installed The project is considered completed once the customer is satisfied that the system runs according to specification

This model is customized for OOD as an evolutionary process of incremental development Emphasis is placed on analysis by the addition of the Architecture Cycle where the top level system software architecture is defined Strong control features have been added to the model by the addition of more control checkpoints

The model is currently being used by various research projects at the University of South Africa Performance figures in terms of cost performance have not been obtained yet Details of the performance of the model may be described in a subsequent paper

5 References

Boehm B W . (1976) "Software engineering", IEEE Trans Computers.

Boehm B W . (1986.) "A spiral model fo software developmentand enhancement", IEEE Transactions on Software Eng Notes, Vol 11, No 4, 1986

Booch G . (1991). "Object-Oriented Design", Benjamin/Cummings.

Coad P . Yourdan E . (1990). "Object Oriented Analysis", Pentice-Hall. 1990

Du Plessis A L . van der Walt E . (1992). Modelling the software development process.
ISCO2 Conference. Egypt.

Graham D R . (1989), Incremental development review of non-monolothic life-cycle development models. Information and Software Technology, vol 31, no 1.

Hofkin B., Powell T.,(1991). Evolutionary Spiral Precess (ESP) Guide Book Version 1. Software Productivity Consortium Quarterly. Summer 1991

Meyer B. (1988). "Object-Oriented Software Construction". Prentice Hall. Hertfordshire

Royce W. W., (1970). "Managing the development of large software systems concepts and techniques". Proceedings. WESTCON. 1979

Rumbaugh J., (1991). Blaha M., Premerlani W. Eddy F. Lorenseb W. "Object-Oriented Modeling and Design". Prentice-Hall International.New Jersey. 1991

Sage A.P. and Palmer J.D., (1990). Software Systems Engineering. John Wiley and Sons. Wiley Interscience Publication. New York.

Yourdon E., (1982). Managing the System Life Cycle a software development methodology overview. Yourdon Press

AN APPROACH FOR INFORMATION SYSTEMS DEVELOPMENT

Talib Damij

University of Ljubljana, 61000 Ljubljana, Kardeljeva pl. 17, Slovenia

Abstract

This approach has three phases. The first phase deals with the problem definition. To gather information about the problem we usually use the interview technique. One way to do this successfuly is by using a very efficient diagram, called user-level entity diagram. The second phase builds the data model of the system using the normalization technique. The third phase deals with a new algorithm to create a complete process model of the system and defines in detail every action which will be realized by the system.

1. INTRODUCTION

In the field of information systems we have many methodologies which may be used to design and implement the desired information system. But whenever I have used any of them, I find it very hard to answer the following question: If we have two or more analysts who are dealing independently with the same problem and using the same method, will they get the same result in the end? The answer to this question is, there is a great possibility that the analysts will create different models for the same information system. The reason for this situation may be found in the fact that almost all of the well known methodologies are depending on the analyst and his/her experience. So the problem may be defined as a lack of these methodologies for leading the analyst efficiently through all phases of information systems development.

This work wants to introduce an efficient approach which assures the guidance of the analyst through all phases of information systems development. The reader will find in this approach a very useful method which minimizes the time needed to information system design and puts the system under a complete control of the analyst through the whole process.

2. PROBLEM DEFINITION

This phase defines the problem to be solved and sets the direction for the whole system. Information must be gathered in an organized way to ensure that nothing is overlooked and that all system detail is captured. To gather information about the system we usually use the interview technique. A well-conducted interview consists of three distinct parts: an opening, a body, and a closing (5). The opening begins by the analyst by introducing his self, the topics he plans to discuss and the purpose of the interview. The analyst must tell the user why he or she was selected for the interview and where appropriate, he must identify the managers who have authorized the interview.

The body must be created carefuly. The analyst should generally begin with a relatively broad question about the user's daily work. Many times we shall notice that the user finds it very difficult to describe his work. Getting the information must be the concern of the analyst. One way to do this successfuly is by using a very efficient diagram, called user-level entity diagram which was developped by Ken Orr (11). In practice these diagrams have been proved that they are very efficient and easy to use by the analyst and also understandable by the user. I am quite sure that they help minizing the time needed to complete this phase.

In the closing of the interview, when the analyst has all information, he must thank the user for cooperating, and offer to make his summary available for review.

2.1. USER-LEVEL ENTITY DIAGRAM

This diagram has to be created by the analyst for each user during the interview and it may represent the result of the interview. The user-level entity diagram is drawn as follows (1):

- Put the organization's name, the user's name, and the system's name at the top of the page.
- Create a bubble in the middle of the page and write the user's name in the bubble.
- Draw in bubbles around the edge of the page and write in the entities (other users) with which the user will interface in this system.
- Draw arrows linking the user in the middle with other entities, showing the interactions beween them.

Let us use a simple example of Account receivable system. The system links the Account receivable section, the Shipping department, the Sales department, and the Customer. Figeres 1, 2, and 3 show the user-level entity diagrams of the above mentioned sections. These entity diagrams may introduce the results of the interviews which have been organized by the analyst with the users in these sections.

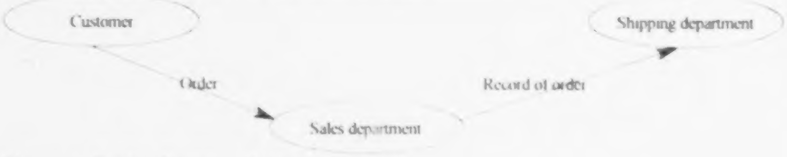


Figure 1. Entity diagram: Sales department

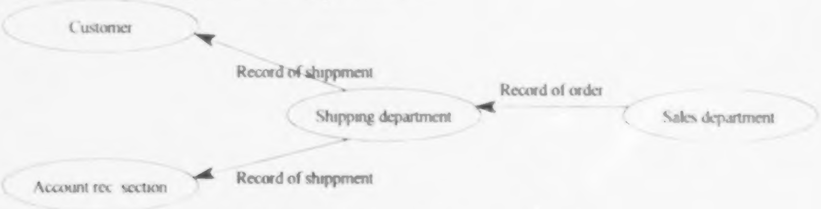


Figure 2. Entity diagram: Shipping department

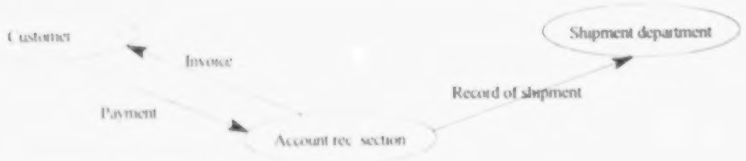


Figure 3. Entity diagram: Account receivable section

The above entity diagrams show exactly which documents are used betwen different entities (users) of the Account receivable system. Existing documents, reports, and displays are important sources of information about the user view and the analyst should collect sample copies of all such documents in the interviews. So, a user view is a subset of data required by a particuler user to make a decision or to carry out some action.

3. DATA MODEL DEVELOPMENT

The second phase deals with the data model development. This is a process of developping a detailed model for the data base of desired information system. This phase has the following three steps: Data modeling, View integration, and Conceptual data model creation.

3.1. DATA MODELING

Data modeling is the process of identifying and structuring the relationships between data elements (9). In this step the analyst normalizes the user views which are identified during the first phase. The result of data modeling is a set of relations in the third normal form for each user view.

3.2. VIEW INTEGRATION

This step is to integrate different sets of relations in one set of relations, which is convenient to all users views. View integration is the process of merging the relations in a single set of relations. These relations represent the conceptual data model for the desired data base expressed in the form of normalized relations.

Two or more relations are merged if they have the same primary key. This means that these relations describe the same entity and may be merged into one relation.

3.3. CONCEPTUAL DATA MODEL DEVELOPMENT

In the previous step the conceptual data model is expressed in the form of normalized relations. For better illustration we transform these relations into a graphical model which clearly shows the relationships and the associations between the relations.

Transformig the relations in our example into a graphical model will result the following conceptual data model:

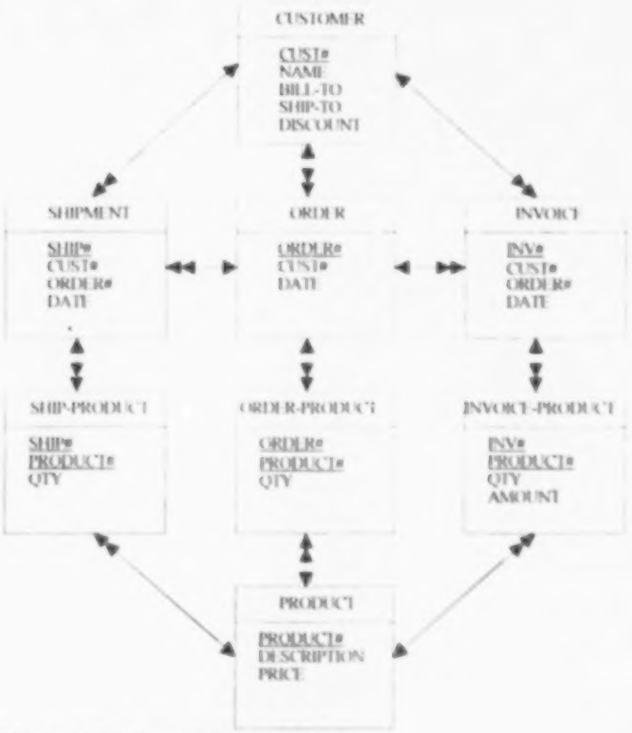

CUSTOMER
CUST#
NAME
BILL-TO
SHIP-TO
DISCOUNT
SHIPMENT
SHIP#
CUST#
ORDER#
DATE
ORDER
ORDER#
CUST#
DATE
INVOICE
INV#
CUST#
ORDER#
DATE
SHIP-PRODUCT
SHIP#
PRODUCT#
QTY
ORDER-PRODUCT
ORDER#
PRODUCT#
QTY
INVOICE-PRODUCT
INV#
PRODUCT#
QTY
AMOUNT
PRODUCT
PRODUCT#
DESCRIPTION
PRICE

Figure 4. Conceptaul data model

4. THE PROCESS MODEL DEVELOPMENT

The result of the first phase is a set of actions or processes which will be realized by the system. And the result of the second phase is a conceptual data model which is ready to be implemented by some DBMS.

The last phase of this approach deals with the process model development. This goal will be achieved in four steps. First step links the entities together to build the system model by using the entity table. Second step links the entities with their actions to group the actions in subsystems by using the entity-action table. Third step puts the actions in some priority order in purpose to concrete their implementation by using the action table. And the last step defines each action or process in detail.

4.1. THE SYSTEM MODEL

The entity table introduces the system's entities which have been recognized in the first phase. The main purpose of this phase is to link entity diagrams together and to create the model for the whole system.

The entity table is created by analyzing the entity diagrams which have been developped in the first phase of this approach. Every entity diagram shows the linkage of different entities by a specified actions. Each action may be determined by two entities:

- The source entity, this is the entity where some action is started, and
- The target entity, this is the entity where the same action is ended.

Corresponding to this idea let us define that the rows of entity table represent the source entities, and the columns of the table represent the target entities. So, each row introduces all actions which are started from a specified source entity defined in this row to other entities of the system. And each column introduces all actions which are reached to a specified target entitiy defined in this column from other entities of the system.

From the first entity diagram in figure 1, we may recognize that we have two source entities, Customer and Sales department, and two target entities, Sales department and Shipping department. To transfer these information to our entity table we have to start by creating two rows for representing two source entities the Customer and the Sales department, and also two columns for respresenting two target entities the Sales department and the Shipping department.

To complete the entity table we have to analyze the entity diagrams one by one and for every action in each entity diagram we must find the specified row and column where the name of the action will be written. The following figure shows the entity table created from entity diagrams in figures 1, 2 and 3.

Source / Target	**Sales department**	**Shipping department**	**Customer**	**Acc. rec. section**
Customer	ORDER			PAYMENT
Sales department		RECORD OF ORDER		
Shipping department			RECORD OF SHIPMENT	RECORD OF SHIPMENT
Acc. rec. section			INVOICE	

Figure 5. Entity table

If in some entity diagram a new source or target entity has been found which is not defined in the table yet, we 'll extend the table for a new row or column depending on the new entity. And also in the case that we have more different actions between the same source and target entities, we 'll write more names of actions in the crossing between them. Forthermore, if the same action is existed in more entity diagrams, we shall ignore it when we 'll realize that an action with the same name has been registered in the table.

When the entity table is completed then we are ready to transfer the information from the table to develop the system model for the desired system. To do so, we must identify the entities which are included in the system (in the rows or in the columns of the table). Any entity in the system model will be represented by a bubble and the name of the entity in it. These entities (bubbles) are connected by the actions between them. Any action is represented by an arrow and the name of the action along it. To connect the entities with their actions, we usually start with first row of the table, fixing the source entity in this row and connecting the bubble of this entity with the bubbles of these target entities which have some action defined in this row.

When the current row is completed then we 'll continue analyzing in the next row. This work will continue until all rows of the table are analyzed. When the analysis of the table is finished, we 'll notice that all information in the table is transfered to the process model which means that the process model for the whole system is developped.

The essence of this step is to create the entity table from different entity diagrams and then using the information collected in the table to create the model of the system. The result of this step is an easy defined and accurate system model which is independent from the analyst and his experience.

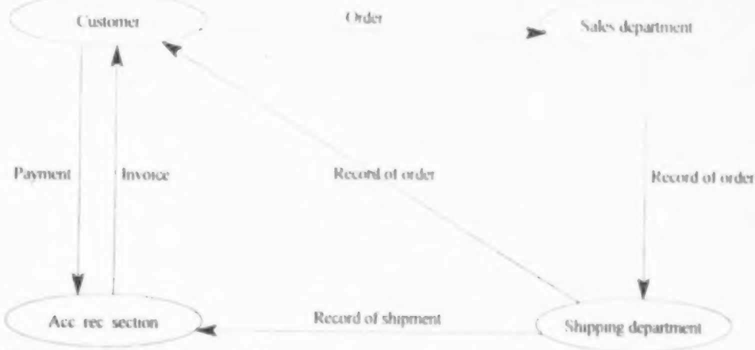


Figure 6. The system model

4.2. THE SUBSYSTEM DEFINITION

The second step of the third phase is dealing with grouping the actions of the system in convenient groups. Every group represents some subsystem in the system. The whole system may be represented by one process. This process is the highest process in the system and called the main process of the system. Main process contains different groups of actions. Every group of actions will represent a subsystem or a process in entity level. So, from the main process we may call any subsystem in the system. Every group of actions or subsystem contains elementary actions or subgroups of actions depending on the subsystem or the requirement in this subsystem.

The best way to create these groups of actions is by identifying the actions which are used by every single entity. To do so we have to create a table which links the entities with their actions. This table is called the entity-action table. Every group of actions will be named by the name of particular entity or by a name which characterizes this group of actions.

The entity-action table shows all actions of the system linked to their entities or users. The entities of the system are represented by the rows of the table and the actions of the system are introduced by the columns of the table. So, any column in the table will show us exactly what is happening with this action defined in this column and which entities are dealing with it. In the other hand, any row of the table will give a clear picture about the actions of a certain entity of the system defined in this row.

Entity Action	ORDER	SHIPMENT	RECORD OF ORDER	RECORD OF SHIPMENT	INVOICE	PAYMENT
Customer						
Sales department	1		1			
Shipping department		1		1		
Acc. rec section					1	1

Figure 7. Entity-Action table.

From this figure we may define the main process of the system, which is represented by the whole table. Forethermore, we may define the subsystems, which are represented by the rows of the table. Every row of the table introduces a subsystem level process, which shows the group of actions defined in this row and linked to the entity of this row. The following figure shows the main process of the system, the processes in the subsystem level, and the processes in the action level.

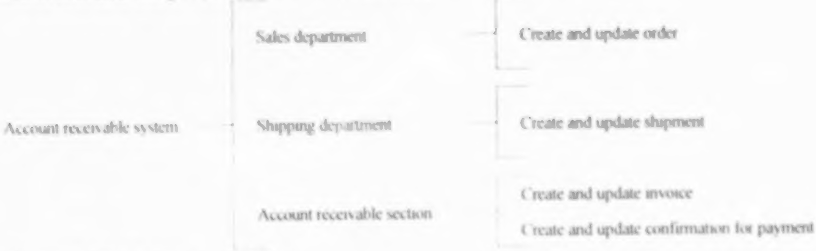

Account receivable system
Sales department
Shipping department
Account receivable section
Create and update order
Create and update shipment
Create and update invoice
Create and update confirmation for payment

Figure 8. The process model

Entity-action table tries to link the result of the system design with the reality of system implemntation. The whole table may be represented by the main process, which is implemented by the main menu of the system. Every row in the table may introduce some subsystem, which is implemented by a certain submenu of the system. Each submenu deals with a group of actions, which may have one or more actions.

This idea will allow the analyst to define system protection very easy by permitting only the authorized users to access to the actions, where they are responsible.

4.3. THE ACTION DEFINITION

The last step of the third phase of this approach deals with defining a priority order for all actions in the system. This order will determine when any of these actions will be developped. This step deals also with creation an algorithm for every action, which describes how to realize this action.

To define the priority order for the actions of the system we use a table called Action table. This table enables us to define a linkage between different actions as in the real world. For a certain action the table will show which other actions have to be realized before this action. In other words, the table will guide the analyst to find out which action must be developped first and which action have to be the second one and so on.

The Action table is structured as follow: All actions are represented in the rows and in the columns of the table. So, every single action occupies one row and one column of the table.

The number of the rows of the table is equal to the number of the actions in the system. The number of the columns is equal to the number of the actions plus one, because the last column is used as priority order column.

To link the actions in the rows with the actions in the columns, we usually register a priority order for every row action starting with the first row and continues row by row, until all rows of the table have been analyzed. For every row in the table, we try to link the action in this row with every action in the columns by finding out if the existence of the action in current column is needed for the development of the action in current row. If the answer is yes then put the priority value of the column action in the crossing between the current row and column, otherwise leave it empty.

To generate the priority order value for every row action we, use the following rules:

- If the current row is empty then the priority order value of the action of this row is zero plus one.
- If the current row has only one value then the priority order value of the action of this row is the value defined in this row plus one.
- If the current row has more values then the priority order value of the action of this row is the sum of these values.

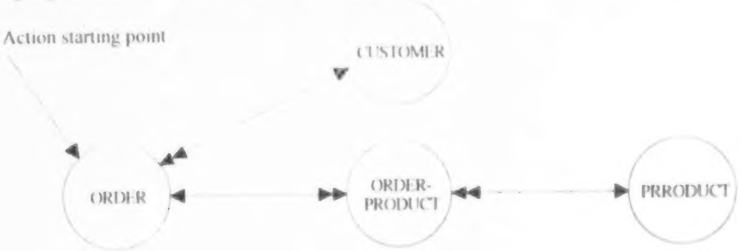
Action / Action	ORDER	SHIPMENT	INVOICE	PAYMENT	Priority order value
ORDER					1
SHIPMENT	1				2
INVOICE	1	2			3
PAYMENT	1	2	3		6

Figure 9. The Action table.

Figure 9 shows the linkage between different actions corresponding to our example the Account receivable system.

When we have finished linking the actions in the rows with the actions in the columns, we'll find a number of values in every row. Using the above defined rules will define a priority order value for every row action. This value will be written in the priority order column of the current row. When we finish creating priority sums for all rows, we 'll see in the priority order column a list of values which represent the priority order for all actions in the table. The action with priority order one is the first action and the analyst has to start realizing this action before any one else. So, the analyst will begin defining algorithms for the implementation of the actions starting with the action with the lowest priority value and continuing to last action with the highest priority value.

When the priority order values of all actions is generated, the analyst has to define every action in detail. To do so, he has to determine two things: the data model segment needed for this action and an algorithm, which describe the realization of the action. The following figure shows the definition of action ORDER:



```
ORDER   READ ORDER
        IF ORDER NOT EXIST
          CREATE A NEW ORDER
        ELSE
          UPDATE ORDER
        ENDIF
```

```
READ CUSTOMER
IF CUSTOMER NOT EXIST
      INSERT A NEW
ENDIF
INSERT A NEW RECORD IN ORDER TABLE
FOR EVERY PRODUCT
      READ PRODUCT
      IF PRODUCT EXISTS
            INSERT A NEW RECORD IN ORDER-PRODUCT TABLE
      ENDIF
ENDFOR
```

```
READ ORDER-PRODUCT
PUT ON SCREEN ALL INFORMATION OF CURRENT ORDER
GET CHANGES
IF CHANGES TO ORDER
      UPDATE ORDER TABLE
ENDIF

IF CHANGES TO ORDERR-PRODUCT
      UPDATE ORDER-PRODUCT TABLE
ENDIF
```

Figure 10. The ORDER action definition.

5. CONCLUSION

This approach has three phases. The first phase defines the problem to be solved and gathers information about the system in organized way using the interview technique. One way to do this successfuly is by using the entity diagrams which was developped by Ken Orr. These diagrams have been proved that they are very efficient and easy to use by the analyst and also understandable by the user. That is why, I am quite sure that they help minimizing the time needed to complete this phase and put the problem under a complete control of the analyst. The result of this phase is a set of actions or goals which will be realized by the system.

The second phase builds the data model.. In this step the analyst normalizes the user views which are identified during the first phase. The result of data modeling is a set of relations in the third normal form for each user view. The analyst continuous with view integration by integrating different sets of relations in one set of relations, which is convenient to all users views. View integration is the process of merging the relations in a single set of relations.

The third phase of this approach deals with the process model development. This goal will be achieved in four steps. The first step links the entities together to build the system model by using the entity table. The second step links the entities with their actions to group the actions in subsystems by using the entity-action table. The third step puts the actions in some priority order in purpose to concrete their implementation by using the action table. And the last step defines each action in detail.

This approach may be introduced as a precise and simple algorithm which defines an order for leading the analyst through all phases of information systems development and that is why it is independent from the analyst and his/her experience.

In the end I would like to say that this approach was used to develop and implement two application systems. First system deals with the plan department in an airways company and the second system deals with diet planning in hospitals. The result of these works is very satisfied.

6. LITRATURE

Connor, D., (1985), "Information System Specification and Road Map", Prentice-Hall, Inc., Englewood Cliffs, New Jersy.

Damij, T., (1989), "Data Base Design", Education center Delta, Ljubljana, Slovenia.

Damij, T., (1992), "An Introduction to Structured Systems Analysis", Informatica, Vol 16, pp. 1 8.

Date C.J., (1977), "An Intriduction to Database Systams", Addison-Wesley Publishing Company, Massachusetts.

Davis W.S., (1984), "Systems Analysis and Design", Addison-Wesely Publishing Company, Massachusetts.

Hawryszkiewycz I.T., (1988), "Introduction to Systems Analysis and Design", Prentice-Hall, Inc., Engleewood Cliffs, New Jersy.

Langefors, B., Verrijn-Stuart, A.A. and Bracchi, G., (1986), " Trends in Information Systems", Elsevier Science Publishers B.V., North-Holland.

Martin, J., and Finclestein, C., (1981), "Information Engineering", Savant Research Studies, Savant Institute.

McFadden, F.R., and Hoffer, J.R., (1985), "Data Base Management", Benjamin/Cummings Publishing Company, California.

Olle T.W., Sol, H.G. and Verrjin-Stuart, A.A., (1982), "Information Systems Design Methodologies: A Comparative Review", Elsevier Science Publishers B.V., North-Holland.

Orr K., (1981), "Structured Requirements Definition", Ken Orr and Associates, Inc., Topeka, Kansas.

Perkinson R.C., (1985), "Data Analysis: The Key to Data Base Design", Wellesley, Massachusets: QED Information Sciences.

SOFT SYSTEMS METHODOLOGY(SSM):A POSSIBLE WAY FORWARD?

Alan Hogarth[1] and John Biggam[1]

[1]Glasgow Caledonian University, Department of Computer Studies, Cowcaddens , Glasgow, Scotland

Abstract

The authors have carried out a survey of five companies The purpose of this survey was to evaluate the use and effectiveness of existing systems development methodologies in the extraction of and specification of user requirements The aim being to identify that within those existing 'hard' and 'soft' methodologies the scope for involving the user in the setting of and understanding of their requirements is deficient As such based on this survey the authors propose a way forward The results of this survey suggest that users are still unsure of how their requirements are extracted and specified

1. INTRODUCTION

Hard Systems Methodologies (HSM) have a long history in academia and in industry and until very recently have been the main development approach employed by practitioners Methodologies such as SSADM (Structured Systems Analysis and Design Methodology),LBMS(1990) claim to tackle the problems of users requirements Soft Systems Methodology(SSM), Checkland (1981), is a different type of methodology, that also claims to tackle the problems of requirements This methodology is now gaining in popularity particularly in regard to those traditionally minded exponents of the Hard Systems Methodologies(HSM) such as Eva(1992) , Ashworth and Slater(1992), who in their books and articles raise the issue of using SSM with hard systems methodologies The authors have selected SSADM and SSM for their survey Four of the companies studied claimed to use SSADM and one claimed to use a 'soft' approach for determining and specifying user requirements Requirements analysis and specification are arguably the two most important aspects of systems development The objectives of systems analysis is to examine all aspects of the system the equipment, personnel, operating conditions, and its internal and external demands, to establish a basis for designing and implementing a better system Failure to elicit a correct and feasible set of requirements from the customer/client at the outset of development can result in the user receiving a final system that bears little resemblance to the original system he envisaged "Requirements are a collection of statements that should describe in a clear,concise and consistent and unambiguous manner what a systems behaviour is to be" Hogarth and Fletcher(1993) This requirements statement should contain enough information to enable the developer to model a systems behaviour that will create a common understanding between customers and analysts. In creating this we must consider the analyst/customer communication and the conflicts that can ensue due to their differing viewpoints.

Many studies have shown that since communication content between analysts and users should be high the chances for misinterpretation and misinformation abound. Communication problems between users and developers of information systems have been recognised as one of the major obstacles in information systems development since at least the late 1960's. Research is still being undertaken in this area indicating that to this day, despite a myriad of methodologies, tools and techniques, the problem of user requirements still appears to remain. It was with due consideration of the aforementioned problems of requirements that author's carried out their survey.

2. SSADM VERSION 4.0 AND THE PROBLEM OF REQUIREMENTS

SSADM Version 4.0 (1990) is a methodology that aspires to tackle the problems of determining and specifying user requirements. It attempts to do this by incorporating two specific "requirements modules". These are Requirements Analysis(RA) and Requirements Specification(RS). Requirements Analysis involves the investigation of the current environment and eventual selection of a Business Systems Option(BSO).

The **Requirements Analysis Module** basically investigates the current environment (mainly processing and data) and considers business system options, i.e.:

Requirements Analysis Module

Stage 1	**Participants**
Investigation of Current Environment	"The investigation team will work to the project manager, and should comprise a senior and experienced analyst, assistant analysts and an active user representative ....." SSADM Version 4(1990)
Stage 2	
Business Systems Options	Requirements analysts with both SSADM and business knowledge,users, IT service providers, staff representatives" SSADM Version 4 (1990)

The above two stages result in the production of various progress reports, catalogues, activity/product descriptions and the selection of a business system option.

A common scenario is that once a project initiative has been deemed feasible, a group of analysts tend to descend on a site and implement an Investigation of the Current Environment producing in the process a wealth of Data Flow Diagrams, Requirements Catalogues, Process Descriptions, Logical Data Structures, etc. Thus from the outset the systems development is **Analyst Driven.** The user is often an onlooker with little input. The main participants are often the analysts with the user merely 'represented' by an in-house member of IT staff.

Requirements Specification Module

The **Requirements Specification Module** can be represented as shown in Figure 1

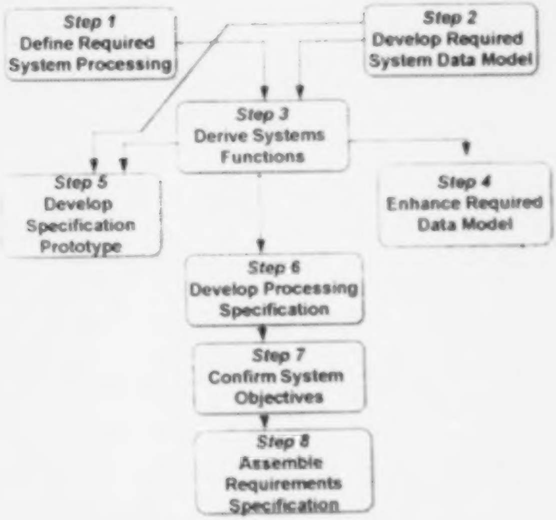

Step 1
Define Required
System Processing
Step 2
Develop Required
System Data Model
Step 3
Derive Systems
Functions
Step 5
Develop
Specification
Prototype
Step 4
Enhance Required
Data Model
Step 6
Develop Processing
Specification
Step 7
Confirm System
Objectives
Step 8
Assemble
Requirements
Specification

Figure 1. Requirements Specification in SSADM V 4

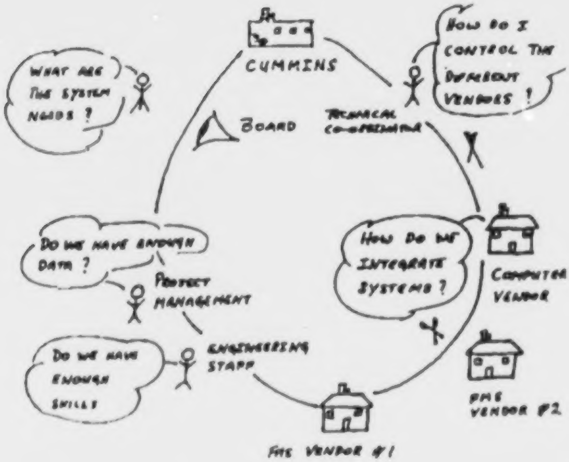
The purpose of this phase is to produce a **Requirements Specification**. After the selection of a business system option, the Definition of Requirements are expanded to finally produce a **Requirements Specification**. The techniques employed involve Data Flow Modelling, Dialogue Design, Entity-Event Modelling, Function Definition, Logical data Modelling, Relational Data Analysis, Requirements Definition and Specification Prototyping. The participants are "Requirements Specification team including data modellers and analysts, functional modellers, entity life history practitioners and other specialists in requirements areas such as capacity planning, security and prototyping."SSADM V 4(1990)

Once again there is an emphasis on analysts and techniques with little attention paid to the views of the user. Indeed, the **Requirements Specification** is 'verified' by the user (normally an IT specialist) only after all the above tasks have been carried out. The **Requirements Specification Module** continues where the **Requirements Analysis Module** left off, and further distances the user from the development of the system by being **Analyst Driven** and placing an emphasis on **completeness** and **consistency** to the detriment of true user understanding of the requirements. Prototyping can also be adopted during this phase and is generally believed to be a good idea because "user understanding is enhanced" . Hogarth and Rao(1993). It needs to be made clear that if the user is rarely involved in the early stages of systems development then what is 'enhanced' is not the user's understanding of his own requirements but the user's understanding of what the analysts believe is the user's requirements.

The interpretation of user participation as an attempt to produce a system appropriate to the needs of the users themselves is not the definition of participation adopted by practitioners of SSADM. SSADM views user participation as an activity in the attempt to produce a system that exhibits **completeness** and **consistency**. This is emphasised by the time allocated to checking and cross-referencing processes, data flows and data stores etc. The purpose of SSADM is to produce a 'correct' system, which is not necessarily the same as producing an appropriate system.

3 SSM AND THE PROBLEM OF REQUIREMENTS

" Problems are perceived as a search for an efficient means of declared objectives or meeting declared needs" Miles (1988). This 'hard' approach was typical of organisations from the 1960's to the 1980's where companies operated as 'goal-seeking' machines and who use information systems to enable the information needs associated with organisational goals to be met. Now there is a noticeable difference in thinking. companies today are more concerned with managing change and it is this ability to operate in rapidly changing markets that is *not* a part of the hard systems approach. Leading edge organisations consider themselves as if they were made of cultures, tribes, political battlegrounds and task networks. Checkland and Scholes(1991). These changes have been acknowledged by newer developments in systems analysis that provide a better approach to tackling the messy, ill-structured problems that characterise human affairs. They define issues rather that problem/solutions, and are better able to examine the strategic implications of organisations. The best known of these emerging approaches is SSM. In SSM the key concept is the need to combine the needs of "natural" (human) and "designed systems" (computers) in to a single human activity system. SSM describes a system according to an individual world view or *weltanschauung* - that allows individual people to hold differing ideas about the system. Graphical methods for naming and modelling *issues* are used to begin the process of modelling in SSM. These models are then compared with real-world actions to tackle the issues. These descriptive models can then be used as a communications tool to structure a debate about change. By the initial use of a *rich picture* SSM and its world-view help to reduce the risk by communicating any potential issues between players in the system. Most importantly it accepts the process of analysis as being part of the system itself. On the rich picture, SSM represents conflicts and other issues, See Figure 2


WHAT ARE THE SYSTEM NEEDS ?
CUMMINS
HOW DO I CONTROL THE DIFFERENT VENDORS !
BOARD
TECHNICAL CO-ORDINATOR
DO WE HAVE ENOUGH DATA ?
PROJECT MANAGEMENT
HOW DO WE INTEGRATE SYSTEMS ?
COMPUTER VENDOR
DO WE HAVE ENOUGH SKILLS
ENGINEERING STAFF
FMS VENDOR #2
FMS VENDOR #1

EXAMPLE RICH PICTURE

People figures:	Represent the key players in the system (analysts as well as clients.)
Cross-swords:	Interfaces and potential areas of conflict
Think bubbles:	Problems to be solved
Big eyes:	Overseeing bodies (usually senior management or standards.)

Figure 2. An example of a *Rich Picture*

The rich picture is then used to determine a *root definition* This expresses the core purpose of the system to be created and is always expressed as a *transformation* process To ensure consistency each root definition is created by considering the elements of a mnemonic - CATWOE, See Figure 3

Definition	**Example**
C - customers or victims of T	- Cummins
A - actors: those who would do T	- Cummins/System vendors staff
T - transformation process: the conversion of input to output	- Greenfield site -> new conn-rod system
W -*Weltanschauung*: the world view which makes the T meaningful in context	- Better performance and greater output
O - owner:those who could stop T	- Cummins
E - environmental constraints: elements outside the system	- Organisational policy

Figure 3: An example of the CATWOE mnemonic

There is much discussion as to how SSM can be used with hard system methodologies to provide a complete system solution SSM appears to have been born out of dissatisfaction with hard systems methodologies The main criticism is that hard systems are too rigid and that the hard solutions are often unsuited to the user's needs There is the essential problem with SSM that if it is not providing solutions, but rather a clarification of the problem-situation, then the problem-situation itself has to be resolved eventually If soft systems are not providing solutions then it logically follows that either hard systems are picking up where the soft systems left off, or a combination of a soft-hard approach will be used to resolve the problem-situation This dilemma has caused some academics to raise questions as to what would happen with merging/embedding soft/hard methodologies Structured Systems Analysis and Design Methodology, and is similarly highlighted by other authors such as (Mingers, 1992 b, Jayaratna, 1992; Paton, 1992; Fitzgerald, 1992, Lewis, 1992)

4. SURVEY SITE AND SAMPLE SELECTION

The sites chosen were five organisations (major airline, legal practice, computer centre, computer manufacturer, catering company) The questionnaires initially concentrate on the study of current systems development approaches (requirements analysis and specification in particular) used in the development of systems and how users/clients perceive these within their organisations **Developers** views on the various problems of elicitation, analysing and specifying requirements are established by means of a questionnaire distributed to five developers with subsequent follow- up interviews In addition 15 **users** (3 from each organisation) are asked to complete a questionnaire relating to how they feel their requirements were represented during the development of their most recent/current system and how they felt they contributed to setting those requirements Follow-up interviews were carried out with two developers and five users One company's user and developer allowed the interview to be taped

5. HSM SURVEY RESULTS

From the analysis of the foregoing questionnaire results and follow-up interviews it was established that four out of the five companies used a structured methodology and they all claimed to use SSADM However during the interviews and further analysis it was found that in all cases SSADM was only partially used In the main the developers had adapted SSADM to suit their particular environment and had merged SSADM with either Jackson's Structured Programming or some sort of 'prototyping' It was also established that most developers (and their superiors) were happy with the results achieved by these methods In analysing the **user** responses a different picture emerges Most users felt that they were involved at an early stage of development in "discussing their problems with the analyst" However during subsequent stages of development most felt that they were ignored with very little consultation with the developer When the developers did discuss the 'requirements' with the users invariably the users were confused by the "charts and diagrams " that were shown to them and they were not aware that a particular development methodology was being used When asked if they felt their original requirements had been met in the production of the final system they answered no However when asked if they were happy with the final system 3 out of 4 replied that they were, in the main, happy with it

The authors found these results illuminating in that they appear to indicate that, although for the past twenty years or more the problem of user requirements have discussed in detail by many authors in the field, the problems are still there particularly when employing a hard approach to systems development Users are still not happy that their needs are being met and feel that they are just not involved enough Furthermore when asked if they would like to participate more in requirements setting they all responded positively It would appear then that the 'hard' approaches , particularly SSADM, are still not doing enough to encourage user involvement in setting their own requirements

6. 'SOFT' APPROACH SURVEY RESULTS

From the results of the questionnaire it was established that one company used a 'soft' approach to systems development The developer did not strictly adhere to all aspects of SSM , but again had adapted it suit his organisation There was more user involvement in discussing the issues that could be involved in finding out about 'problem situations' A 'rich picture' was used "He drew a diagram with funny figures to describe the workings of the office" the user questionnaires indicated a positive response to this method They felt they had been consulted and that they had a clear idea of what their problems were and what they required to eliminate these problems They also felt the at the developer was easily accessible if they wanted to ask him something When both developer and users were asked about the level of user involvement at the various stages of development they responded with a 'medium' and 'high' answer One interesting point to note was the fact that the users did not feel their requirements had been met in the final system although again as in the 'hard' approach results they were , in the main, happy with the final system In the subsequent interview it transpires that although they felt they were involved, they did get confused sometimes with what the analyst was saying to them and did not want to "appear foolish" by telling him that they did not understand him They also expressed some doubt as to whether the approach adopted was substantial enough for the system they wanted They felt that although the method did highlight many problems , it didn't really offer a solution for them After the discussions "when it became technical" they did not like to interfere to much

The results here appear to indicate that although the users were happy with the increased involvement in setting their requirements they felt that a 'real' solution was not forthcoming and in the end their requirements had not been met The results shown by both 'hard' and 'soft' users indicating that they were relatively happy with the final system,although their requirements had not been met,indicates some confusion in the users perception of what their requirements are when asked about this in the interview the response was generally that " the system looked good" or "it was better than before" This would indicate that there is still some work to be done to increase the communication and understanding between users and developers

7. THE WAY FORWARD

The results indicate that there is a need for a methodology that can incorporate the effective parts of a hard approach like SSADM and incorporating a 'soft' approach such as SSM and introducing quality aspects Consistency and completeness can enhance a quality system but in themselves they do not guarantee quality

Quality is increasingly becoming an issue in most business environments (witness the growth in BS5750 !) We shall define quality as follows

Quality = In conformance with user requirements

i e **Quality = Fitness for use**

A quality system is therefore a system that meets the needs of the user This means that the user must be a participant throughout the life of a systems project i e from Project Initiation to Analysis to Design through to Implementation and to Training and beyond Clearly, then, the early stages of systems development are vital to the future success of a project if the needs of the users are determined and agreed by all relevant parties - in the form of a Requirements Specification - at an early stage then there is a greater chance of the system gaining acceptance at the implementation stage

A quality system necessarily involves the users and as such a type of Soft Systems approach is inevitable However a system does require to be documented and SSADM does consist of well tried and tested methods of documentation which, as already, stated do meet the requirements for verification, validation, completeness and consistency Finally quality procedures must be incorporated into any quality system The SSADM model would remain but supplemented by a "Soft Systems approach" and incorporating "quality procedures"

CONCLUSION

In this paper the authors discussed, based on the results of their survey, how users appear to perceive the use of requirements analysis and specification methods in the setting of their requirements Prior to this discussion the problems of requirements were considered with reference to both 'hard' approaches (SSADM) and the 'soft' approaches (SSM) and how they claim to tackle these problems The results of the survey appear to indicate that the problems of requirements still remain particularly in the areas of user understanding and communication with the developers Based on these findings The authors initially propose that a possible way forward may be to incorporate SSADM with SSM and include certain 'quality procedures'

In summary we consider the following premises

1. that **completeness** and **consistency** enhance a **Quality System** In themselves they do not guarantee quality, but they are necessary attributes

2. a **Quality System** necessarily involves the users Thus a type of Soft Systems approach is inevitable

3. the system requires to be documented SSADM consists of well tried and tested methods of documentation which also meet the criteria for completeness and consistency

4. **Quality Procedures** require to be incorporated into any **Quality System** (these are introduced into the **Requirements Analysis Module** and the **Requirements Specification Module** of SSADM Version 4)

References

Ashworth,C and Slater,I(1992). "An Introduction to SSADM Version 4 ",Mcgraw-Hill,Maidenhead

Checkland,P (1981). "Systems Thinking Systems Practice",Wiley,Chichester

Checkland, P and Scholes,J (1991),"Soft Systems Methodology in Action" Chichester, Wiley

Eva, M(1991),"SSADM Version 4 A User's Guide",Mcgraw-Hill,Maidenhead

Fitzgerald,(1992) ,"On linking soft systems methods and information systems methods", Systemist, Vol14, No 3 p126

Hogarth ,A and Fletcher, I."The essence and accidents of requirements engineering", Proceedings of the 9th National Conf on Manufactturing Research, Bath, 1993

Hogarth,A and Rao, G,"Focus on requirements engineering in SSADM Version 4", Proceedings of the 1st national Conf on Information Systems Methodologies, BCS Specialist Group, Edinburgh, 1993

Jayaratna, N (1992), "Should we link SSM with information systems!", Systemist, Vol 14, No 3,pp108-119

LBMS/CCTA *SSADM Version 4. Manual* - London 1990

LBMS/CCTA *SSADM Version4. Manual* ,pp RA-SM-9- London 1990

LBMS/CCTA *SSADM Version 4. Manual* ,pp RA-SM-9- London 1990

LBMS/CCTA *SSADM Version 4. Manual*,pp RA-SM-5 -London 1990

Lewis, P (1992), "The feasibility and desirability of linking SSM and data-focused, information systems development", Systemist Workshop

Miles, R K(1988),"Combining soft and hard systems practice grafting or embedding?", Journal of Applied Systems Analysis, 15, pp55-60

Mingers, (1992),"SSM and information systems an overview", Systemist, Vol 14, No 3,pp82-88

Paton, G(1992), "Using SSM for information systems", Systemist, Vol 14,No 3,pp120-122

The application of Systemic Risk Analysis techniques to project management methodologies.

Roger W. Stewart, School of Information Systems, Kingston University, Penrhyn Road, Kingston upon Thames, Surrey, KT1 2EE, England.

Abstract

Undertaking an Information Systems project is a high risk activity. The ability to identify potential risks and to take steps to avoid them are two of the most important aspects of good project management. Non-systemic techniques for risk evaluation and management such as net present value calculations, Monte Carlo simulation and contingency planning, are already widely applied and understood, but the use of of systems approaches in this context is less widespread. It can be argued, however, that by using systems approaches at appropriate stages in project management and risk analysis methodologies it is possible to identify potential risks right at the start of the project life cycle and that their use at the other end of the cycle can enable lessons from outcomes to be used to improve performance on future projects and facilitate organisational learning. This paper concentrates on the scoping phase and the use of systemic analysis techniques in order to identify potential risk areas.

1. INTRODUCTION

IS project managers today are faced with ever-increasing complexity. This is due in no small part to the fast changing business environment in which projects are typically carried out, but it has also been brought about by the changing nature of projects themselves. The single large project was the traditional domain of IT project managers, but projects now come in all shapes and sizes. The advent of the project-oriented organisation, matrix-managed projects, networked projects, CASE projects, rapid development projects, and organisationational change projects have all changed the scope of what is now termed a 'project'. Furthermore, this change of focus has been accompanied by greater requirements for project managers to demonstrate management expertise. The traditional skills of a project manager such as risk analysis, estimation, work breakdown and the use of critical path networks must now be enhanced by those concerned with strategic analysis, general management and human relations. Evidence of these changes can be seen in the "holistic, multi-related and multi-dimensional Concept and Framework of Project-Management (the integrative model)" of Saynisch (1990), and the development of the field of systemic-evolutionary project management.

Systems methods and techniques for describing areas under investigation and for modelling alternative futures are well established in many other fields. Through the use rich pictures, systems maps, influence diagrams, systems models and the like, holistic pictures can be built up that emphasise interconnectedness. These in turn can enable problem themes to be identified. By perceiving situations (real and potential) as systems, the way in which components relate to each other and the relationships between systems and their environments can also be investigated.

In the IS project management context such methods allow the standard parameters of a project - aims, inputs, planned activities, outputs, timescales and so on - to be examined. But perhaps more importantly where risk management is concerned, they enable two further areas to be investigated:

1. Interactions, such as those between the project and its operating climate and wider environment, and those within the project team and between the team and its clients. These can frequently be even more important than the individual components of the project.

2. Human aspects, such as conflict of objectives, motivation problems, poor communication and so on, which may in themselves threaten the success of the project.

Examples of the use of systems methods and techniques in IS project management context are given in the paper, together with illustrations of how such analyses can be used in the identification of risks.

2. PROJECT LIFE-CYCLES

Risk identification and the development and implementation of risk management strategies must be carried out throughout the life of a project and as Berkley, Humphreys and Thomas (1991) comment: "The aim is to encourage project managers to be sensitive to these potential sources of risk, to be able to anticipate their occurrence, to appreciate their potential impacts on the project objectives and to reduce their future impact through appropriate risk action management strategies."

However, the earlier risks are predicted and the wider the area that is searched to identify them the better the chance of developing effective contingency plans to deal with them. Figure 1 takes Yeates's (1991) typical life cycle of an IT project and overlays it with the risk phases to which systems techniques can be applied.

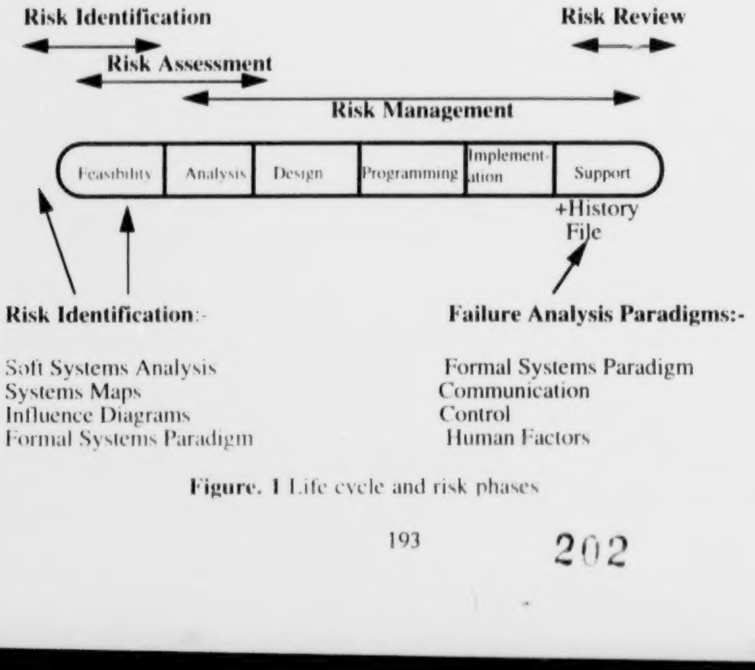

Risk Identification
Risk Review
Risk Assessment
Risk Management
Feasibility
Analysis
Design
Programming
Implementation
Support
+History File
Risk Identification:-
Soft Systems Analysis
Systems Maps
Influence Diagrams
Formal Systems Paradigm
Failure Analysis Paradigms:-
Formal Systems Paradigm
Communication
Control
Human Factors

Figure. 1 Life cycle and risk phases

3. PROJECT SCOPING

The scoping of a project enables a project manager to understand the objectives and deliverables of the project and thus to encapsulate what needs to be managed. In essence it defines the inputs, processes and the intended outputs. Traditionally it has been undertaken using techniques such as cost benefit analysis, NPV and the calculation of predicted returns on investment. Other techniques such as the identification of critical success factors, PEST analysis and stakeholder analysis are increasingly being used to supplement these traditional methods but essentially all of these are reductionist techniques and thus do not allow all of the uncertainty surrounding projects to be examined. Furthermore, they ignore the interaction between disparate risks. To cope with the increasing complexity identified earlier in this paper an holistic approach is also needed. To borrow a phrase from Gareis (1990), 'holistic problem viewpoints' are neccessary in order to understand the nature of modern complex projects.

3.1 Project Management Methodologies

Several IS project management methodologies explicitly contain risk identification and management stages. Systemic analysis techniques can provide valuable holistic information to improve the effectiveness of these, for example:

PRINCE (PRojects IN Controlled Environments).
A methodology from the CCTA in widespread use that provides a framework within which projects can be correctly specified, designed and implemented. Although a risk analysis methodology does not yet form part of the overall methodology, the analysis of risk is seen to be an essential part. Risks being assessed at the start of a project and at the end of each control stage. The method adopted is the use of a checklist of the various elements contributing to risk and a calculation of the likelyhood of the risks occurring, their effect and in adopting any contingency plans. Systemic analysis can provide risk information about the project and its environment to aid the construction of the checklist, and for evaluating the impact of risk areas already defined.

RISKMAN (The European project Risk Management Methodology).
This was developed by a European consortium under the Eureka research programme and is in the process of being released. The methodology encompasses various risk identification, quantification, analysis and mitigation techniques throughout the project management process. Systemic analysis of both business, human factors and the interrelationships between them can provide assistance in the identification of risks and in the quantification stage.

Systems methods and techniques are, by definition, holistic in nature and thus able to make good the deficiences of analyses based solely on reductionist thinking. They are complementary to existing methods involving the use of experienced intuitive management and experts, standard questionaires and checklists, structured interviews and the use of expert computer-based systems. Application of a range of systems methods and techniques to project management will now be considered. The design and development of an airpline front desk check-in system will be used to provide examples.

3.2 Holistic Techniques

Soft Systems Analysis:-

SSA is an approach that deals not only with hard tangible information, such as the structure of an area under analysis and facts/figures, but also with the soft complexity that arises because people are or will be involved. It takes account of the feelings, attitudes, perceptions and so on of individuals and groups and enables potential conflict between people to be taken on board. (Use of the full soft systems approach in the context of project management is explored by Stewart and Saunders (1990).

One aspect of SSA that is particularly relevant to the identification of risk is a problem analysis technique called rich picture analysis which allows problem themes to be identified. A rich picture is an holistic representation of the situation as seen by an analyst. It incorporates structure (how things fit together), processes (what happens) and human 'soft' elements (attitudes, feelings, emotions, conflict etc.). Problem areas or themes can then be identified from the picture. These may be tangible, eg financial, technical or regulatory constraints, or intangible as in the case of human feelings and actual/potential conflict. Figure 2 shows a rich picture of an airline front desk IT project.

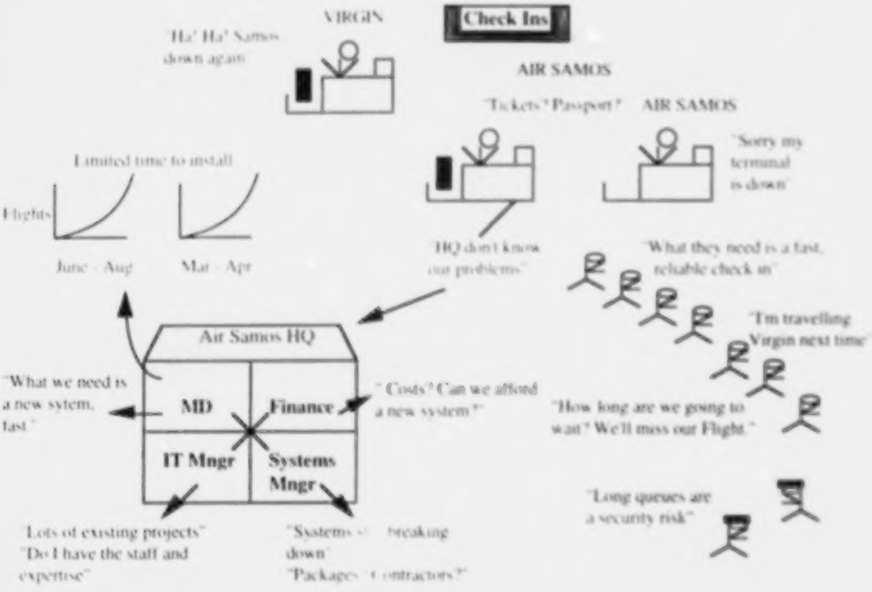

VIRGIN
Check Ins
'Ha' Ha' Samos down again'
AIR SAMOS
'Tickets? Passport?'
AIR SAMOS
'Sorry my terminal is down'
Limited time to install
Flights
June - Aug
Mar - Apr
'HQ don't know our problems"
"What they need is a fast, reliable check in"
"I'm travelling Virgin next time"
Air Samos HQ
"What we need is a new sytem, fast."
MD
Finance
" Costs? Can we afford a new system?"
IT Mngr
Systems Mngr
"How long are we going to wait? We'll miss our Flight."
"Long queues are a security risk"
'Lots of existing projects"
"Do I have the staff and expertise"
"Systems are breaking down"
"Packages / contractors?"

Figure 2. Rich Picture of the airline front-desk project.

Examples of problem themes that can be identified from Figure 2 are:

Financial - such as funding and development costs, return on investment and inflation.
Physical - timing of installation eg. peak holiday period, development time, staff loading.
Stakeholders - such links to security, passengers, computer staff, finance.
Technology - new hardware, communications links, staff existing skill levels.

These problem themes represent risks that will need to be managed. In addition they can be used as inputs into other forms of analysis such as PEST, the evaluation of the political, environmental, social and technical factors that have an effect on the project.

A complex project scenario is likely to contain within it many different groupings of people, organisations and factors that could be regarded as 'systems' (sets of interconnected components that do something). A system can be viewed as an abstract notion of a whole, as in SSA, or, as in other approaches, as the analyst's perception of an apparently useful collection of real world entities. Under either, interpretation systems can be used as a basis with which to explore structure, processes, interconnectedness, emergence, and so on, of the area under investigation. The decision of which systems to investigate should be informed by knowledge of the project situation and its history and take account of the different viewpoints and perspectives of those involved in it.

Systems diagramming techniques provide a means of modelling the chosen system(s) and thus of investigating the problem area. Two examples will be given here: the systems map, which concentrates on the structure of the proposed project; and the influence diagram, which emphasises relationships and interconnectedness.

Systems Maps:-

A systems map is essentially a diagram showing a snapshot of the structure of the area under consideration. It specifies the components of a system and its environment as seen by the analyst at a specific point in time. An example based on the airline front desk project is shown in Figure 3.

This diagram is essentially a broad brush representation of those elements that are deemed important to the project, drawn at the start of the project. Components that are included within the boundary are those which together are perceived to form the front-desk development system. The elements which are outside the boundary are those which will affect the performance of the system but are not neccessarily affected by it. The positioning of the boundary may change as more information is gained from analysis and as the project proceeds.

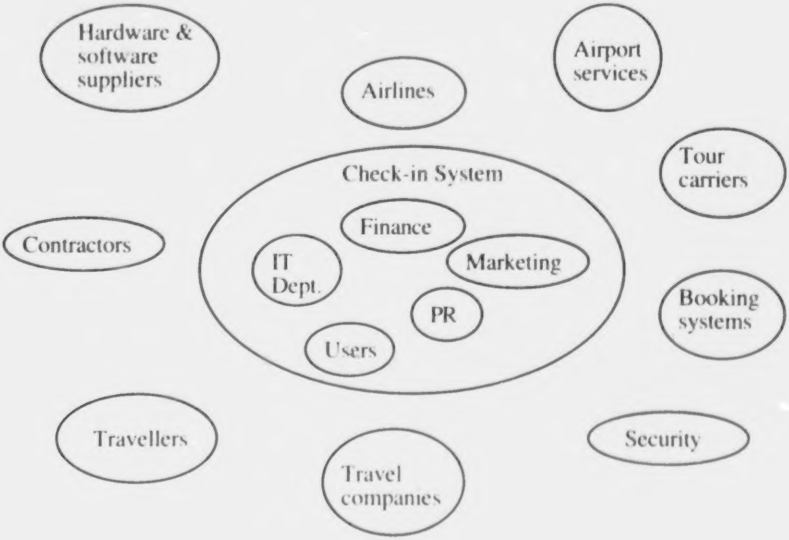


Figure 3. Systems Map of Check-in System

The systems map indicates the main structural features of the project and its environment. As such it can be used in the scoping process to provide useful information as to who are the key players in the project; for example, where resources come from (suppliers, eg. hardware/software supliers and contractors), and where constraints may emanate (current staffing and skill levels, volume of travellers, links to outside systems, security etc.). At a high level it indicates the actual and potential stakeholders, (travellers, check-in staff, PR, Finance etc.), whose expectations of what constitutes success for the project must be considered. All of these components need to be looked at to identify and subsequently evaluate the risks involved. Having identified components within the system and its environment it is important to look at the interactions between them, a systems map can be further developed to produce an influence diagram.

Influence Diagrams:-

An influence diagram is also a snapshot of the main structural features of the area under consideration, but in addition it explores the important relationships between components within the system and between the system and its environment. Distinctions can be drawn between different types of influence, such as influence via financial sources, existing staff loading, project timing, output and so on. On the diagram they can be shown by spatial relationships and by the use of different types of lines and arrows. Figure 4 gives an example of the components that influence each other. It can be further developed to show the strength and nature of the influence.

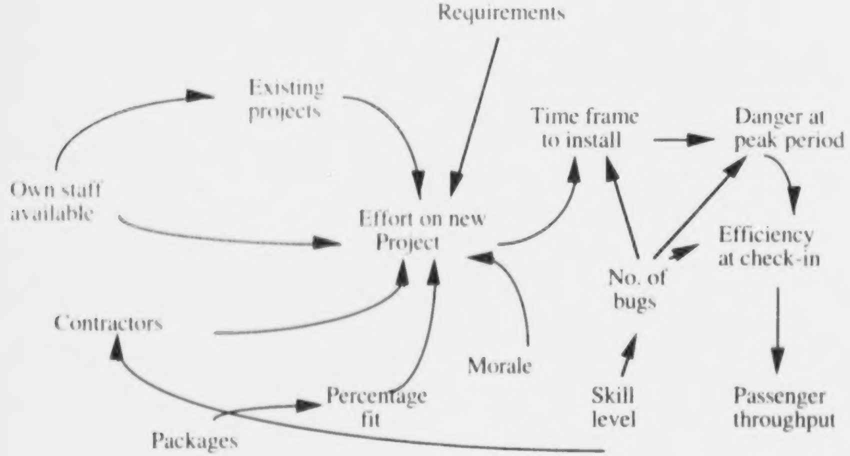


Figure 4. Influence Map of the Check-in project

The influence diagram can be used by the project manager in several ways. For example it can show key areas that will influence the project length, or the potential effect of a series of interacting components, such as packages with a good fit may reduce the effort on the new project which can have an effect on the development and installation time frame and through the chain to passenger throughput. The strength of influences can provide inputs into a critical success factor analysis, for example the strong influence of existing skill levels on the quality of the end system. Each of the influences identifies potential risks which need to be considered and managed.

The preceding two diagrams have shown for the area under consideration the structural components and their influences and how it can apply to risk analysis. The project manager can utilise a futher model, the Formal System Paradigm, to identify where risk factors can be seen.

Formal System Paradigm:-

The formal system paradigm (FSP) coordinates a number of key systems concepts within an organised framework. It is adapted from Checkland (1979) and is essentially a compilation of those features which are necessary if a set of activities is to comprise a system capable of purposeful activity without failure. The features encompassed within it are:
a continuous purpose or mission;
a measure of performance;
a decision taking process;
components which are themselves systems;

a degree of connectivity between components;
an environment within which the systems interact;
boundaries separating the system from its wider system and the wider system from the environment;
resources;

The FSP is shown diagrammatically in Figure 5. Further information about its development and use can be found in Fortune and Peters (1990), Fortune (1993).

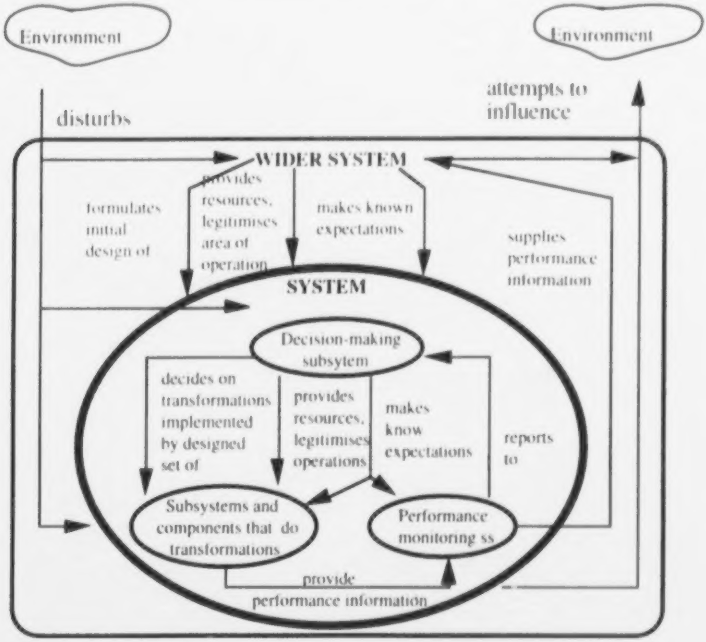

Environment
Environment
disturbs
attempts to influence
WIDER SYSTEM
formulates initial design of
provides resources, legitimises area of operation
makes known expectations
supplies performance information
SYSTEM
Decision-making subsytem
decides on transformations implemented by designed set of
provides resources, legitimises operations
makes know expectations
reports to
Subsystems and components that do transformations
Performance monitoring ss
provide performance information

Figure 5. Formal Systems Paradigm

By using the FSP at the start of a project it is possible to gain insights into potential failures and hence into the risks that may threaten the project. It provides a standard against which the set up and management of the project can be judged. Any discrepancies that emerge from comparing the project with the FSP can be fed forward to modify the project plans or prepare contingecy plans. Experience has shown the following to be typical discrepancies and hence risks that were not identified and managed.

Deficiences in the apparent organisational structure of a project, eg non-involvement of stakeholders such as end-users

Deficiences in links, eg. no clear statement of purpose supplied in comprehensive form from the wider system (makes known expectations), or within the project the correct allocation of resources (internal and external staff)

Deficiences in the performance of one or more of subsystems, eg. inadequate performance measurement subsystem, inadequate estimation, incorrect work breakdown etc.

BEST COPY AVAILABLE

Subsystems with ineffective means of communicating to other subsystems,eg *vague reporting* to the project board (PRINCE), and no use of the Risk Register on a continuous re-assessment basis (RISKMAN).

It can be seen that comparison of the actual situation to the FSP will show discrepancies which need to be addressed in structuring the project and areas that are potential risks needing to be managed

4. Conclusions

This paper has adopted the position of Gareis (1990), that 'holistic viewpoints' are a neccessary part of modern complex project management. Systems approaches are by definition holistic in nature and examples of techniques used in these approaches have been examined in the context of two project management methodologies. It has been further argued that emphasis on the systemic should begin as early in the project as possible. The paper has demonstrated how a project manager can use systems diagramming and a formal model to assist in the scoping of a project whereby the proposed project itself is examined and its relationship to its environment is investigated to identify and highlight potential risk areas.

5. References

Berkley, D., Humphreys, P.C., and Thomas, R.D., 1991, Project risk action, management, *Construction Management and Economics*, Vol. 9, E. & F.N. Spon Ltd.

Bentley, C., 1992, "Introducing PRINCE", NCC Blackwell.

Carter, C., et al, 1994, "Introducing RISKMAN", NCC Blackwell.

Checkland, P. B., (1979), "Systems Thinking, Systems Practice", Wiley, Chichester.

Fortune, J., (1993), Systems Paradigms, The Open University Press, Milton Keynes, UK

Fortune, J., (1993), "Studying Systems Failures", The Open University Press, Milton Keynes, UK.

Fortune, J. and Peters, G., (1990), The formal system paradigm for studying failures, *Technology Analysis and Strategic Management*, 2,4 p383- 390.

Gareis, R., 1990, Management by Projects, Proceedings vol.1, 10th International Congress on Project Management, MANZ Verlag, Vienna.

Saynisch, M., 1990, Project Management in relation of a changing understanding of management and social systems, Proceedings vol.2, 10th International Congress on Project Management, MANZ Verlag, Vienna.

Stewart, R.W. and Saunders, R.G., 1990, New Perspectives in Project Management in Research and Development, Proceedings vol.2, 10th International Congress on Project Management, MANZ Verlag, Vienna.

Yeates, D. A., 1991, "Project Management for Information Systems", Pitman.

OBJECT ORIENTED DEVELOPMENT PARADIGM AND ITS DELIVERABLES

Wita Wojtkowski and W. Gregory Wojtkowski

Computer Information Systems and Production and Operations Management
Boise State University, 1910 University Drive, Boise, Idaho 83725, USA

Abstract

In addition to analysis, design and construction steps, any computer information systems applications development comprises also coordination, and evolution. All construction steps demand work segmentation as well as appropriate reuse and management of deliverables. Object orientation is currently much discussed systems development paradigm- It introduces not only new tools but also new deliverables and activities, and is apt to profoundly affect applications development. This paper discusses aspects of object orientation that especially impact the deliverables of the applications development process.

1. INTRODUCTION

Object oriented development is characterized (between other things) (Jacobson et al., 1992; Martin and Odell, 1992) by:

- incremental and iterative development which demands that modeling the problem domain is done throughout application development

- encapsulation of process with data in both the application structure and the development methodology

- anthropomorphic design in which objects in the application are assigned "responsibilities" to carry out
- emphasis on reuse and extensibility of both, the design and code

Incremental and iterative development are not new: they did not originate with object orientation. They are well established in modern, non-object oriented methodologies (Bohem, 1988). Only extensibility actually relies on object oriented mechanism of inheritance (Snyder, 1993). Nevertheless, each mechanism provided by object orientation offers systems developers a new way of thinking about the system decomposition as well as construction (Embley et al., 1992).

Most importantly, object orientation offers multiple new mechanisms for reuse. Although it may complicate decisions about which to use, it allows (and even might enforce) significant creation of reuse products as well as consumption of reuse (Johnson and Foote, 1988; Griss and Tracz, 1993, Wojtkowski, 1994). For example, in the context of reuse, classes and frameworks (Snyder, 1993) are applications development units that can be treated as organizational assets to be Cultura(and protected).

In this paper we examine the impact of object orientation on applications development, focusing especially on the mechanisms that aid reuse and extensibility.

2. DIMENSIONS OF APPLICATION DEVELOPMENT

There are many dimensions to applications development (Sommerville, 1992). In order to focus this discussion, we discern three dimensions:

1. *construction* that includes deliverables, actions and tools;
2. *coordination*, that includes scheduling and work segmentation;
3. *evolution* that includes reusel

Our contention is that object orientation can affect many aspects of applications development: the deliverables produced, sharing and controlling the evolution of deliverables, actions, techniques and tools used, and the scheduling strategy and the segmentation of work.

2.1 Construction

Construction involves not only final deliverable (the application), but also models of the application at various levels of detail, such as requirements, design descriptions, test cases, and consequently numerous intermittent deliverables. Each of these deliverables is described using a notation. Tools are employed to work with each notation. Team Produools might be specialized and automated (for example CASE) or comprise only pencil and paper.

Thus any consideration of deliverables is naturally combined with judgment on notation and tools. Since object orientation produces different deliverables, the notation and tools are also different. Thus we can conclude that once the deliverables and their notations change, the entire construction portion of the methodology might be expected to change. Let us consider simple illustration of the above point: In the average object oriented tool kit, the code browser facilitates the management of multitudes of small (typically 3 to 10 lines long) subroutines. Coding habits for such small subroutines might be different than those for subroutines averaging 30 to 50 lines.

2.2 Coordination

A natural way to segment is by deliverables. For a project that requires more than one person, both, the deliverables and work elements must be segmented out. These segmentation naturally calls for staging and scheduling of deliverables. Traditional methods such as waterfall development carry the restriction that all requirements be defined, reviewed and approved before high-level design or analysis can begin (Sommervillei 1992). In turn, high-level design must be completed before component design can commence, and so on. Alternatives to the waterfall strategy are incremental and iterative strategies inherent in object oriented approaches (Embley et al.t 1992).

We assert, however, that the *incremental* staging and scheduling strategies of object oriented methodologies are basically the same as those of non-object oriented ones. In order to examine this assertion we must be mindful about the terms *incremental* and *iterative*[1]. In support of the assertion we offer these interpretations: 1) Iterative development supports *predicted* rework of portions of the system. Its intent is to allow correction of mistakes and product improvement, based on, for example, the user feedback and performance tuning, and to allow this in a controlled manner (Figure 1).

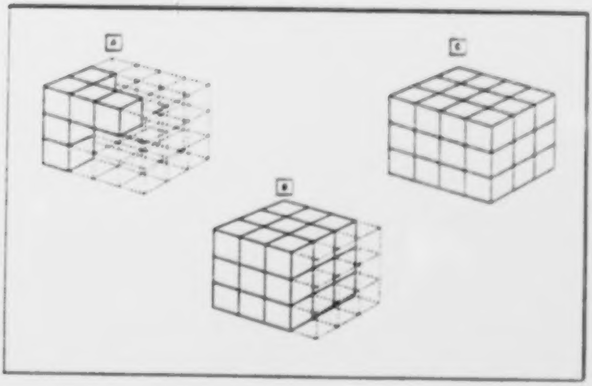


Figure 1. Iterative Development Model

2) Incremental development supports development of portions of the system *at different times or rates*, these portions to be integrated into the system as they are completed. Its intent is to develop a system piece by piece, to allow additions to, for example, requirements, improvements to the development process, or changes to scheduling (Figure 2).

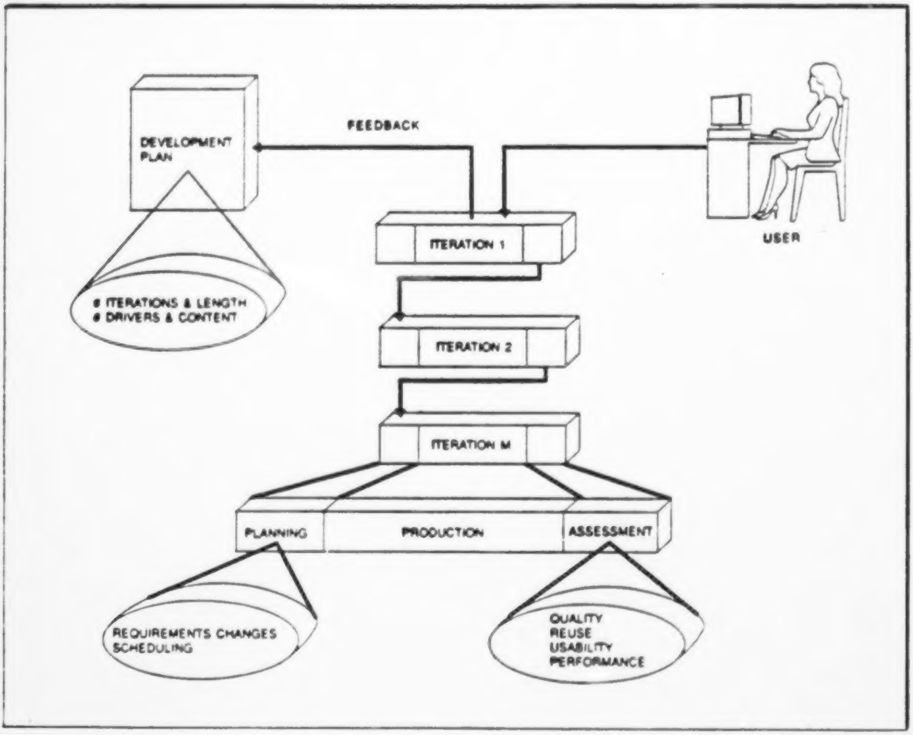

DEVELOPMENT PLAN
FEEDBACK
USER
# ITERATIONS & LENGTH
# DRIVERS & CONTENT
ITERATION 1
ITERATION 2
ITERATION M
PLANNING
PRODUCTION
ASSESSMENT
REQUIREMENTS CHANGES
SCHEDULING
QUALITY
REUSE
USABILITY
PERFORMANCE

Figure 2. Incremental development model

The essential characteristic of the incremental development is that the system is developed in portions. As the portions are completed they are added to a growing system. The increments may or may not be staged in parallel (Pittman, 1993). The essential characteristic of iterative development, on the other hand is the act of discovery of new information on, for example, requirements and thus on improvement of system design.

The development project may be both, incremental and iterative. It also may be conducted with or without iterative development. In incremental development project without iterations, for example, the increments are developed to full production standards from the start.

2.3 Evolution

Evolution of the items of object oriented development can be classified as single point or multiple points evolution. Single point of evolution is present when a change to an item is automatically reflected in all occurrences of its use. Multiple points of evolution are present when a change to all occurrences of an item can only be accomplished by making changes in multiple places.

Object orientation provides three new mechanisms for reuse and extension: classes, inheritance, and polymorphism. Most object oriented methodologies also include recommendations for sharing class definitions and objects. Moreover, extensibility relies upon reuse. It involves creating a new solution from an existing one by programming the difference.

Any reuse can be classified as clean or messy. Clean reuse is reuse in which the reused item needs no retesting (new combination of parts will need testing, though). Messy reuse is reuse in which retesting of the reused item is required.

When the reuse is being tracked within one system, tools can be applied to examine the entire system to locate all references to an item in question. (And to construct both forward and backward traces.) The class hierarchy browser serves as an example of such a tool.

3. IMPLICATIONS

Nothing in the definition of object orientation indicates that any particular staging and scheduling strategy ought to be adopted: both incremental and iterative development strategies are already standard recommendations in the non object oriented methodologies. Nevertheless, object orientation truly facilitates incremental and iterative development.

Parallelism in development due to use of frameworks tas prestructured designso is the biggest contribution of object orientationbeframeworks provide a vehicle givformalization and replication of some of the previous design decisions. The structure provided by a framework is likely to be a destillation of previous, successful designs.

Objects, classes and frameworks can be considered organizational assets in the same way as are database definitions and data. The general technique for managing the evolution of objects, classes and frameworks is thus the same.

When serious attention is given to reuse, there are positive and negative aspects to consider. Positive aspects are embodied in that reusing previous solutions takes considerably less time than construction from scratch. Negative aspects are that when reusing, designers and programmers are not actively designing and programming. Thus in the context of reuse, productivity of a programmer , for example, ought not to be measured in lines-of-code produced (this work against reuse), but as the rate of achieving reuse. Since object orientation brings with it multiple new mechanisms for reuse it also forces rethinking of the notions of productivity.

4. REFERENCES

Bohem, B., (1988) A spiral model of software development," *Computer*, Vol.: 21, pp. 61-72.

Embley, D., Kurtz, B., and Woodfield, S., (1992), "Object-Oriented System Analysis and Specification: A Model-Driven Approach," Prentice-Hall, Englewood Cliffs, NJ.

Griss, M. L., and Tracz W., (1993) Workshop on software reuse, *ACM Software Engineering Notes*, Vol. 18, pp. 74-85.

Jacobson, I, Christerson, M, Jonsson, P, and Overgaard G., (1992) "Object-Oriented Software Engineering," Addison-Wesley Publishing, Wokingham, England.

Johnson, R., and Footie, B., (1988) Designing reusable classes, *Journal of Object Oriented Programming*," Vol. 1, pp. 22-35.

Martin , J. and Odell, J., (1992) "Object-Oriented Analysis and Design," Prentice-Hall, Englewood Cliffs, NJ.

Sommerville, I. (1992), "Software Engineering", Addison-Wesley, Wokingham, England

Snyder, A., (1993) The essence of objects: concepts and terms, *IEEE Software*, Vol. 10, pp. 31-43.

Pittman, M., (1993) Lessons learned in managing object oriented development, *IEEE Software*, Vol. 10, No. 1, pp. 43-54.

Wojtkowski, W. (1994) Democratization of information systems automation through reuse, forthcoming in *Proceedings of 3rd International Conference on Systems Integration*

A LANGUAGE AND TOOL FOR THE ENGINEERING OF SITUATIONAL METHODS FOR INFORMATION SYSTEMS DEVELOPMENT

Frank Harmsen, Sjaak Brinkkemper, Han Oei

Centre for Telematics and Information Technology, University of Twente, P.O. Box 217,
7500 AE Enschede, the Netherlands, E-mail {harmsen, sjbr, oei}@cs.utwente.nl

Abstract

This paper describes, starting from an information systems development project situation, the concept of configuring situational, or project-specific, methods for information systems development. This concept is called Situational Method Engineering. We outline the steps to be taken in Situational Method Engineering. Moreover, various types of method building blocks, so-called method fragments, are described. These method fragments are represented both graphically and by means of the language **MEL**. We show **MEL** operations for the adaptation, selection, and manipulation of method fragments. An architecture for a support tool for Situational Method Engineering is presented.

Keywords: Information systems development, Method Engineering, Situational Methods, Computer Aided Method Engineering

1. INTRODUCTION

The process of developing and adapting information systems is, after more than 25 years of Software Engineering, still subject to a variety of problems. In the past decade, these problems have been attempted to tackle by proposing a number of standard Information Systems Development Methods. Such an Information Systems Development Method (ISDM), being a collection of procedures, techniques, tools, and documentation aids which will help the systems developers in their effort to implement a new information system, should ensure a controlled, standardised engineering approach to information systems development.

However, every information systems development project is unique with respect to, for instance, its mission and its organisational setting. This is the reason why the application of standard ISDMs does not guarantee successful IS projects. Due to the fact that a standard ISDM is inflexible, i.e. does not match exactly any project situation, it is adapted in an ad-hoc fashion before each single project. We illustrate this with the following case, which is based on (Olle, 1988) and which will be referred to throughout this paper.

The existing inventory control and purchasing information system of the XYZ company has to be converted from COBOL to a 4GL environment. Moreover, a *knowledge component has to be built in, because the future information system has to anticipate intelligently and flexibly to changed inventory levels in relation to seasonal influences. The existing stock control database has to be converted and extended. The management of XYZ initiates the project CAI to develop the required system. This project is performed and managed by "Advice" software developers company and can be shortly characterised by: conversion 3GL -> 4GL with knowledge component and database conversion.*

The project manager has to make a plan for the project, which is to be based on a well-known ISDM, say Information Engineering (IE) (Martin, 1989). However, this method does not suit the project situation, because the project requires, compared to IE and the CASE tool being used, some extra steps, products, and tools. For instance, Information Engineering does not provide the specific steps to design a knowledge based system. Moreover, a lot of activities, for instance the entire Information Systems Planning stage, and products of IE seem to be unnecessary, and products have to be adapted in order to comply with the in-house standards of XYZ.

So, the project manager needs to tune IE to the project situation. He removes steps and deliverables of IE, and adjusts some product descriptions of IE to the company standards. In order to build the knowledge component, he provides the method with some steps and deliverables of the knowledge systems development method KADS (see for instance (Schreiber, 1993)).

The result of this process is a project approach. But is it also a situational *method*, in other words, does the approach meet the consistency and completeness requirements of a true method? The project manager has anticipated flexibly to the project situation, but are quality control, efficiency and standardisation, being only three of the advantages of the employment of a standard ISDM, not at stake?

We claim that the process of adapting a standard method has to be executed in a controlled and efficient way. It has to be provided with a set of method fragments, which are chosen and assembled using a set of pre-defined guide-lines, and it has to be supported by a computerised tool. In this way, a balance can be found between the required flexibility and the rigidity of standard ISDMs. We indicate this balance by the term "Controlled Flexibility", which is to be achieved by using a so-called *Situational Method Engineering* approach (Harmsen, 1994). Situational Method Engineering is a sub-area of *method engineering* (Kumar, 1992; Heym, 1993; Slooten, 1993), which we define as the engineering discipline to design, construct and adapt Information Systems Development Methods, including CASE tools. Situational Method Engineering focuses on the design, construction, and adaptatation of *situational methods*, which are ISDMs tuned to the situation of the project at hand. Situational Method Engineering provides standardised building blocks and guide-lines, so-called meta-methods, to assemble these building blocks, in order to yield situational methods.

Situational Method Engineering has, besides the fact that every project situation is different, several motivations. One reason is, that large projects, and with them their methodological requirements, evolve, so that the method employed has to be adapted during the project. Another reason is, that there will be always new emergent paradigms and technology break-throughs, for which new methods have to be defined. The study of Hong et al. (Hong, 1993) shows that large parts of the new methods are taken from other methods and techniques. Situational Method Engineering provides an efficient and proven way to anticipate on, and to stimulate new technologies, by accelerating the method development process.

The importance of situational method was already recognised in (Olle, 1991). This "scenario philosophy" has been further elaborated by Kumar and Welke, who introduced Methodology Engineering, being an approach to develop and implement methods (Kumar, 1992). A method representation model providing ISDM concepts and a technique to analyse and compare existing methods was presented in (Heym, 1993). Saeki et al. described also a method representation model, as well as a data base called Method Base from which several complete ISDMs can be selected (Saeki, 1993). In (Hidding, 1993) the notion of task package, being a part of the process perspective of methods, is introduced. Van Slooten et al. outline the construction process of situational methods, emphasizing the determination of the project characterisation (Slooten, 1994). In (Harmsen, 1994), the structure of a method base to be filled with parts of existing ISDMs, called method fragments , is presented.

This paper is based on (Harmsen, 1994), and focuses on means to represent, select, and manipulate method fragments. It is organised as follows. After the introduction, we will describe in section two the Situational Method Engineering process and concepts. Section three pays attention to the graphical representation of method fragments, whereas section four focuses on the method representation and manipulation language **MEL**. Section five outlines the required automated support for Situational Method Engineering. The paper ends with conclusions and a description of further research.

2. SITUATIONAL METHOD ENGINEERING

The steps to be taken in configuring a situational method are depicted in figure 2.1. It is assumed that "Advice" possesses a method base, filled with parts of existing methods, techniques and tools covering a large number of situations. The method fragments in the method base are inserted and updated by a methods administration function, after they have been selected and represented. The configuration of situational methods is performed by a method engineer.

Starting point of this Situational Method Engineering process is the CAI project environment, which includes the existing information infrastructure, the users, the organisational culture of both "Advice" and the XYZ company. Project factors (also called contingency factors), such as the required availability of an AI component, the fact that conversion constitutes a large part of the project, and the available development expertise, are determined forming a project characterisation.

The project characterisation is input to the process of selecting suitable method fragments from the method base. Method fragments include activity descriptions, product descriptions and tools, for project execution as well as project management. In the CAI project, at least method fragments to perform a conversion from COBOL to a 4GL environment, method fragments to design an AI component, and method fragments to convert the database are needed. The choice of the method fragments requires validation with respect to the project environment.

Method fragments are assembled into a situational method, a consistent ISDM tailored to the CAI project. The method is applied in the project, where it may turn out to be necessary to refine or adapt the method. This may be due to the changing project environment or to the clarification of the project characterisation, i.e. project contingencies that were not clear before can now be rated. In order to facilitate information systems development knowledge accumulation, experiences with respect to the situational method should be kept, and are therefore, after an evaluation, stored in the method base.

The process outlined above constitutes the basis of a meta-method, a method to develop -situational- methods. Besides the fact that the steps can be performed in various orders, different philosophies can be adopted while configuring a situational method. To mention just a few: the process can be product-oriented or process-oriented, can be performed top-down or bottom up, or can be performed with fragments originating from only one ISDM or from more ISDMs (Harmsen, 1994).

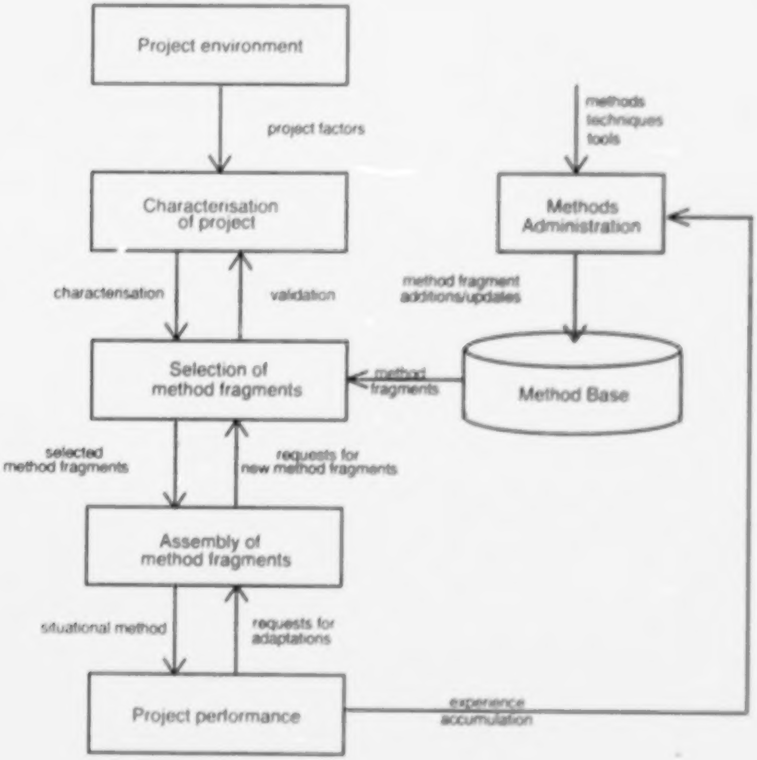

Project environment
project factors
methods
techniques
tools
Characterisation of project
Methods Administration
characterisation
validation
method fragment additions/updates
Selection of method fragments
method fragments
Method Base
selected method fragments
requests for new method fragments
Assembly of method fragments
situational method
requests for adaptations
Project performance
experience accumulation

Figure 2.1 The process of configuration of situational methods

3. METHOD FRAGMENTS

In order to organise the method base and to facilitate method construction, we classify method fragments along the three dimensions *perspective*, *abstraction*, and *granularity level*.

The perspective dimension constitutes of the *product* perspective and the *process* perspective. Product fragments represent deliverables, milestone documents, models, diagrams, etc. Process fragments represent the stages, activities and tasks to be carried out. Figure 3.1 depicts an example of a process fragment.

The abstraction dimension constitutes of the *external level*, the *conceptual level*, and the *technical level*. Method fragments on the conceptual level are objective descriptions of information systems development methods or part thereof. Figures 3.1 an 3.2 are examples of such descriptions. External method fragments are introduced to accommodate multiple views on methods (see also (Wijers, 1991; Finkelstein, 1992; Sowa, 1992; Baldwin, 1993)). Important views to be distinguished are: the method engineer's view, considering the method as a goal in itself, the project managers's view, considering the method as a means to control the project, and the developer's view, considering the method as a set of guide-lines to construct an information system. Technical method fragments are implementable specifications of the operational parts of a method, i..e. the tools. Some conceptual fragments are to be supported by tools, and must therefore be accompanied by corresponding technical fragments. One conceptual method fragment can be related to several external and technical method fragments.

The granularity level dimension possesses in principle an infinite number of elements, i.e. granularity levels. A granularity level of a method fragment can be imagined of as the level in the decomposition tree of a method on which the method fragment resides, with the root of the method, i.e. the entire method, as level 1.

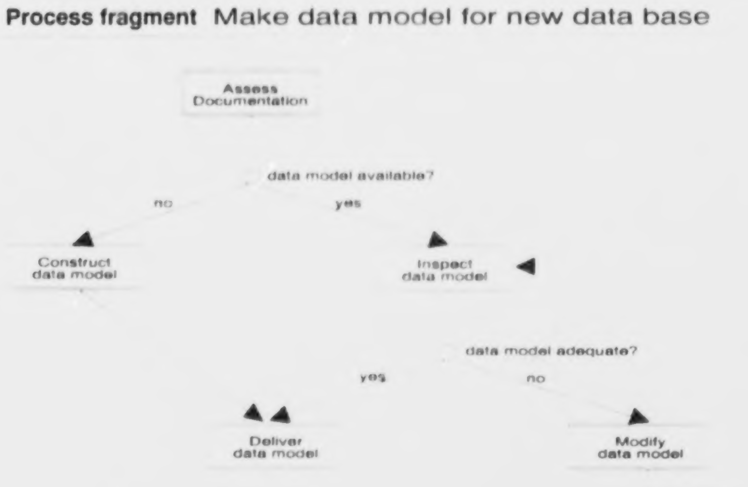

Process fragment Make data model for new data base
Assess Documentation
data model available?
no
yes
Construct data model
Inspect data model
data model adequate?
yes
no
Deliver data model
Modify data model

Figure 3.1 Process Structure Diagram of a process fragment

4. A METHOD ENGINEERING LANGUAGE

For a compact representation of method fragments and operations on method fragments, we introduce the language **MEL. MEL** is a method representation, manipulation and query language, and is used in conjunction with graphical modelling techniques and mathematical formalisms.

Figure 4.1 shows two **MEL** descriptions, one of which is graphically depicted in figure 3.1.

```
CONCEPTUAL PRODUCT Simple ERD:

(
   CONCEPT Entity;
   CONCEPT Relationship;

   ASSOCIATION involves
   (  INVERSE is involved in;
      FROM Relationship TO Entity;
      CARDINALITY (1,2;1,m)
   )

   SOURCE Amethod
   GOAL Data modelling
)
```

```
CONCEPTUAL PROCESS Make data
model for new data base:
(  ACTIVITY Assess Documentation;
   DECISION data model available?
   (IF yes
      ACTIVITY Inspect data model;
      DECISION data model adequate?
      (IF yes
        ACTIVITY Deliver data model
       IF no
        REPEAT Inspect data model)
   IF no
      ACTIVITY Construct data model;
      ACTIVITY Deliver data model)).
```

Figure 4.1 Method fragment descriptions in **MEL**

To let the methods administrator adapt method fragments descriptions, for instance the addition of the concept *Attribute* and a corresponding association to the simple binary ERD modelling technique, we use **MEL** as a method administration language:

```
Add to Simple ERD CONCEPT Attribute
Add to Simple ERD ASSOCIATION has(INVERSE belongs to; FROM Entity TO Attribute;
CARDINALITY (0,m;1,m)))
```

To enable the method engineer searching for the right method fragments, for instance all product fragments of IE , **MEL** is used in the following manner as a query language:

```
Select PRODUCT
Where SOURCE = IE, PURPOSE = Data_model, EXPERIENCE = High.
```

To assemble the selected method fragments, **MEL** is used as a method manipulation language:

```
ASSOCIATION describes
(  INVERSE is described by
   FROM Entity TO Data Store
   CARDINALITY (0,m;1,m)
)
Join Simple ERD With DFD Through ASSOCIATION describes
```

Note that **MEL** can be used in two ways: as a method administration language by providing the means to represent and manipulate the internal structure of method fragments, and as a method assembly language, by providing language constructs to select and manipulate method fragments. An example of a -simplified- view on the resulting situational method, described in **MEL**, is shown in figure 4.2.

```
PROJECT CAI - Inventory Control System XYZ:
(
  ACTIVITY        - Business Area Analysis;
                - Knowledge Elicitation and Modelling;
                - Business System Design
                - Technical Design;
                - Realisation and Testing;
                - Data Conversion;
                - Installation.
  ).
```

Figure 4.2 Overview of the situational method for CAI project

More comprehensive and formal descriptions of syntax and semantics of **MEL** can be found in a forthcoming paper.

5. COMPUTER AIDED METHOD ENGINEERING

Since designing a method is, like information systems design, an information creation and transformation process, Situational Method Engineering has a lot in common with Information Systems Engineering. Similarities do not stop at the computer aided support for the activities involved. Due to the same reasons why Information Systems Engineering should be supported by CASE tools (McClure, 1989), Situational Method Engineering should always be applied in conjunction with supportive tools, called Computer Aided Method Engineering (CAME) tools. An effective CAME tool should provide support for the Situational Method Engineering steps we outlined in section two.

Currently, we are developing Decamerone, a CAME tool that is based on and is used in conjunction with the meta-CASE tool Maestro II (Merbeth, 1991; Harmsen, 1993; Harmsen, 1994). A meta-CASE tool is a fully adaptable CASE tool. The architecture of Decamerone is depicted in figure 5.1.Arrows in this figure depict data flows, rounded rectangles depict data stores and boxes depict Method Engineering support functionality.

Method fragments, including method outlines, are stored in the method base. Fragments are inserted, modified, or deleted through the methods administration component, which can be perfomed graphically or through **MEL** specifications. The method assembly component is used for the selection, by means of dialog boxes or **MEL** queries, and assembly of fragments from the method base. It provides also an interface through which the project situation is characterised, and it provides method consistency check facilities.

The method assembly component results are stored in a CAME-internal ISM (Intermediate Situational Method) data base. This Intermediate Situational Method is processed by several generators, which provide the meta-CASE tool with the Situational Method data. These data include a project repository, a work-breakdown structure, project planning data, a set of diagram- and text editors, and help screens and paper manuals. The meta-CASE tool needs this input in order to fulfil its role as a CASE tool during the IS project. The approach that views both tools and methods as method fragments results in integrated CASE tools, i.e. covering the complete project life cycle, that are interwoven with the ISDM.

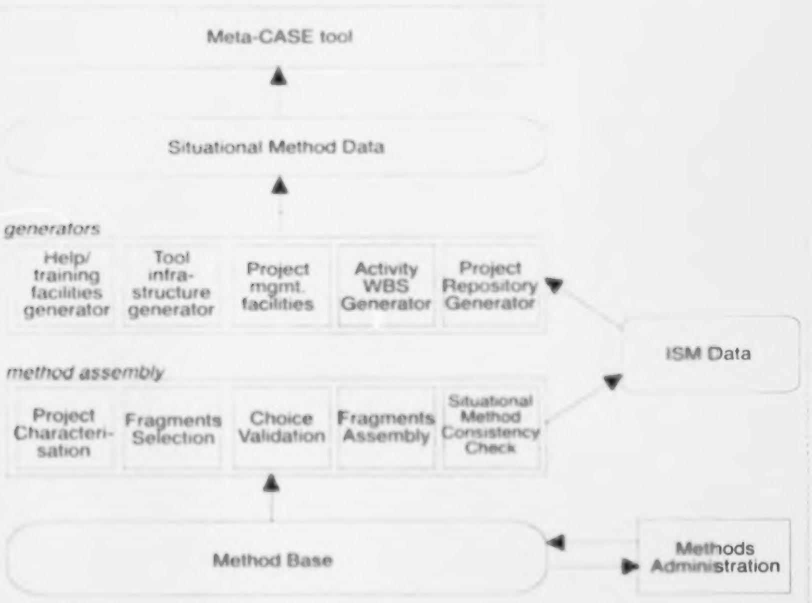

Meta-CASE tool
Situational Method Data
generators
Help/ training facilities generator
Tool infra-structure generator
Project mgmt. facilities
Activity WBS Generator
Project Repository Generator
ISM Data
method assembly
Project Characteri-sation
Fragments Selection
Choice Validation
Fragments Assembly
Situational Method Consistency Check
Method Base
Methods Administration

Figure 5.1 Architecture of Decamerone

6. CONCLUSIONS AND FURTHER RESEARCH

Illustrated with a practical case, we have introduced the notion of Situational Method Engineering. We have shown the process of configuring situational methods, as well as the different types of method fragments with which this process takes place. Different approaches to Situational Method Engineering have been outlined. Method fragments are represented, adapted, selected, and manipulated by means of the Method Engineering Language **MEL**, of which examples have been provided.

Situational Method Engineering is a young and promising research area. Topics for further research include the classification of contingency factors, heuristics to select suitable method fragments, and the accumulation of project experience in the method base. Our research focuses on guide-lines for method construction, specification and implementation of a method base, and specification and implementation of a CAME tool.

The structure and contents of the method base need further refinement by means of formalisation and tests on all kinds of methods, preferably from diverse sources and with different underlying paradigms. An additional problem to tackle are the ways to fragment a method.

Prototypes of the method base and CAME tool are available at the moment. These prototypes will be further refined and tested in the systems development practise.

Empirical research into the practice of method building and method tuning is needed to obtain more consistency rules. Consistency rules will be formalised as complete as possible, in order to allow for an easy incorporation into the CAME tool.

REFERENCES

Baldwin, D., (1993), Applying Multiple Views to Information Systems: A Preliminary Framework. In: Data Base, vol. 24, no. 3, pp. 15-30.

Finkelstein, A., J. Kramer, B. Nuseibeh, (1992), Viewpoints: A Framework for Integrating Multiple Perspectives in System Development. In: International Journal of Software Engineering and Knowledge Engineering, Vol. 2, No. 1, pp. 31-57.

Harmsen, F. and S. Brinkkemper, (1993), Computer Aided Method Engineering based on Existing Meta-CASE Technology. In: Brinkkemper, S. and F. Harmsen (Eds.), Proceedings of the Fourth Workshop on The Next Generation of CASE Tools, Paris.

Harmsen, F., S. Brinkkemper, H. Oei, (1994), Situational Method Engineering for Information System Projects. To appear in: Olle, T.W., A.A. Verrijn Stuart, J.L.G. Dietz (Eds.), Proceedings of the IFIP WG8.1 Working Conference CRIS'94, Maastricht.

Heym, M., and H. Österle, (1993), Computer-aided methodology engineering. In: Information and Software Technology, vol.35, no. 6/7, pp 345-354.

Hidding, G.J., G.M. Freund, J.K. Joseph, (1993), Modeling Large Processes with Task Packages. Workshop on Modeling in the Large, AAAI Conference, Washington, D.C.

Hong, S., G. van den Goor, S. Brinkkemper, (1993), A Comparison of Object-Oriented Analysis and Design Methodologies. In: Proceedings of the 26th Hawaiian Conference on System Sciences (HICSS-26), IEEE Computer Science Press.

Kumar, K. and R.J. Welke, (1992), Methodology Engineering: A Proposal for Situation-Specific Methodology Construction. In: W.W. Cotterman, J.A. Senn (Eds.), Challenges and Strategies for Research in Systems Development, Wiley.

Martin, J., (1989), Information Engineering (3 vols.), Prentice-Hall, Englewood Cliffs.

McClure, C.L., (1989), CASE is Software Automation, Prentice-Hall, Englewood Cliffs.

Merbeth, G., (1991), Maestro II - the integrated CASE system of Softlab (in German: Maestro II - das integrierte CASE-System von Softlab). In: Balzert, H. (Ed.), CASE Systeme und Werkzeuge, 3e Auflage, BI Wissenschaftsverlag.

Olle, T.W., (1988), Business Analysis and System Design Specifications for an Inventory Control and Purchasing System. In: Olle, T.W., A.A. Verrijn-Stuart, L. Bhabuta (Eds.), Computerized Assistance during the Information Systems Life Cycle, North-Holland.

Olle, T.W., J. Hagelstein, I.G. MacDonald, C. Rolland, H.G. Sol, F.J.M. van Assche, A.A. Verrijn-Stuart, (1991), Information Systems Methodologies - A Framework for Understanding, 2nd edition, Addison-Wesley.

Saeki, M., K. Iguchi, K. Wen-Yin, M. Shinohara, (1993), A Meta-Model for Representing Software Specification & Design Methods. In: N. Prakash, C. Rolland, B. Pernici (Eds.), Proceedings of the IFIP WG8.1 Conference on Information Systems Development Process, Como.

Schreiber, G., B. Wielinga, J. Breuker (Eds.), (1993), KADS: A principles approach to knowledge-based system development, Academic Press, London.

Slooten, K. van, and S. Brinkkemper, (1993), A Method Engineering Approach to Information Systems Development. In: N. Prakash, C. Rolland, B. Pernici (Eds.), Proceedings of the IFIP WG8.1 Conference on Information Systems Development Process, Como.

Slooten, K. van, S. Brinkkemper, P. Hoving, (1994), Contingency Based Situational Systems Development in Large Organizations. To appear in: Khosrowpour, M. (Ed.), Proceedings of the 1994 Information Resources Management Association International Conference, Idea Press.

Sowa, J.F., and J.A. Zachman, (1992), Extending and formalizing the framework for information systems architecture. In: IBM Systems Journal, vol. 31, no. 3, pp. 590-616.

Wijers, G.M., (1991), Modelling support in information systems development, Dissertation University of Delft, Thesis Publishers, Amsterdam.

DEVELOPING SOFTWARE CLOSE TO ENVIRONMENT: A SITUATIONAL APPROACH

Maya Daneva

Institute of Mathematics, 1113 Sofia, Ac. Bontchev str., bl.8, Bulgaria

Abstract

This paper deals with the problem of how to close the software development process to the environmental changes. We present a situational interpretation of software product development. A general characterisation of a situation is provided and a situational approach to software development is outlined.

We also discuss the process of modelling the software life cycle and the documentation development.

1. INTRODUCTION

To produce high-quality software with a planned amount of resources and to a predicted schedule, the project management should monitor many environmental factors impacting on the development process. This forces the software engineers to model the software life cycle in dependence on the dynamics of the environment. Generally, the well known software development models emphasise the problem of assessing management risk items: the changes in the initial requirements (Krasner, 1989), the project costs (Boehm, 1988), the new problems to be solved (Wild and Maly, 1992), etc. None of these models takes the positive effect of the environmental changes into account.

In this paper, we present a situational perspective to software development. We argue for a situational approach which considers not only a particular factor, but a wide range of both positive and negative, controllable and uncontrollable factors. All environmental factors are amalgamated into a situation specific for a certain time during the software development process. Thus, our objective is to trace continuously the software development by means of a sequence of situations and to keep the development process corresponding to the environmental changes.

We also propose a situational software life cycle model and a way to develop the documentation.

2. TO CHARACTERISE A SITUATION

2.1. The notion of situation

To propose a definition for the item "situation", we take the following meaning of the term as a starting point: any situation amalgamates the factors impacting positively or negatively on the software development process at a given time. This interpretation corresponds to the general meaning of the term. For example, Webster's New World Dictionary (Third College Edition) reports four meanings for the word "situation".

- Manner in which a thing is situated in relation to its surroundings.
- Position or condition with respect to circumstances.
- A combination of circumstances at any given time.
- Difficult or critical state of affairs.

In our paper, to identify a situation is to determine a set of factors (e.g. the circumstances) relevant to software development at a certain time and to evaluate factors' influence on the development process. We give a more precise definition for the term "situation":

Situation is defined as a set of factors making up the environment at a given time. The environment here means the possibilities the developers profit from and the constraints and rules they need to follow.

2.2. Classes of environmental factors making up a situation

The factors considered in a given situation can be grouped in eight classes:

1. Macro-environmental factor class. It concerns environmental monitoring, e.g. the activities to anticipate changes in external environment (economic, legal, political, social and cultural).

2. Competitive factor class. It refers to competitive monitoring. This class can include the following factors: the extent of the software industry rivalry, the competitors' strategy, resources and marketing mix, etc.

3. Technological factor class. It involves all factors influencing on the relationship among the information technology, information system and information management (Earl, 1988), i.e. the factors which drive, shape and control the information technology infrastructure.

4. Financial factor class. It is related to the investments, the accounting policies and the budget schedule.

5. Organisational factor class. It concerns the organisational behaviour and the organisational effectiveness of the software development staff.

6. Software Quality factor class. It refers to the common key attributes (maintainability, reliability, efficiency and userfriendness) which the well engineered software product should exhibit (Sommervile, 1992), as well as to the specific software features (functionality, performance, quality of documentation, reusability, interoperatebility, service and support).

7. Software Engineering factor class. It concerns all tools, techniques and approaches applied to requirement engineering, specification, implementation, testing, etc.

8. Marketing factor class. It consists of the factors impacting on marketing strategy development, marketing audit, marketing objectives, marketing mix decisions.

For clarity, we consider the factor classes separately, although their effects on the software company are most often combined.

Next, all identified factors can be divided into two groups in dependence on the developers' ability to control them. Thus, we distinguish between controllable and uncontrollable factors. The factors of the first group are directly or primarily under the control of the developers. In contrast, the uncontrollable factors present forces over which the developers' control is restricted or completely impossible.

Furthermore, to describe a certain situation we identify the essential factors of each class and we determine to which extent these factors can be controlled.

2.3. Building the profile of a situation

Let St be a situation at the time t. Let the set F={F1,...,Fn} represent the factors amalgamated into the situation St, e.g. the elements of the set F are considered as attributes of the situation St. Thus, the situation St can be presented on the factors (attributes) of the set F. Each factor is weighted against its importance to the software development. Let W={W1,...,Wn} be a vector each element Wj of which represents the weight of the factor Fj, j=1,...,n. We assume that the weights are normalised, i.e.

$$\sum_{i=1}^{n} Wi=1$$

Let the factor classes be denoted as Class1,...,Class8. The elements of the set F can be of different classes. An proper scale should be selected to assess the influence of each factor on the development process. To determine the complexity of the situation St with respect to a certain class Classi, i=1,...,8, we introduce the so called Partial Complexity Rate (PCR). It is calculated as follows:

$$PCR(Classi) = \sum_{k=1}^{s} (Fk*Wk)$$

where Classi, i=1,...,8 is a factor class, s is the number of factors of this class, fk is the influence evaluation of the factor Fk and wk is the factor weight.

On the base of the PCRs we could build a profile of the situation St (Figure.1.) and compare it against the profiles of other situations encountered. To present the situation's profile, we use a two-dimensional space where X axis represents the PCR and Y axis - the classes.

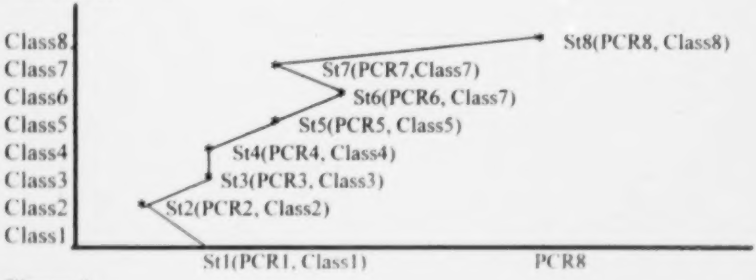


Figure 1

The point Sti(PCRi,Classi), i=1,...,8, means the PCR of the situation St with respect to the class Classi. The picture constructed by connecting all points Sti, i=1,...,8 represents the situation's profile. The latter is aimed to be used in comparing situations. By analysing profiles we determine the factors which affect most frequently changes in software development and we clarify the nature of the so called dominant problem, i.e. we identify what kind of problem the company actually has. We assume that the dominant problem is related to the factor class with respect to which the situation has the highest PCR. For example, if the PCR(Class8) is the highest among all PCRs, than one can generally conclude that the dominant problem the company meets is a marketing one.

The dominant problem changes during the software development, i.e. each situation may state a specific problem. The relation between the factor class that causes the dominant problem and the time the situation occurs is presented in Table1. where the first column contains the factor classes and the next columns - the data of the situation occurrence.

Table1

	date1	date2	date3	date4	date5	date6
Class8	x					
Class7				x		
Class6			x			
Class5					x	
Class4		x				x
Class3						
Class2						
Class1						

The cell (Class*k*, date*i*) marked by "x", k=1,...,8, i=1,...,n means that at the time datei the developers meet a dominant problem affected by the factor class Classk. After analysing and comparing all situations occurred one is able to draw conclusions about the nature of the dominant problem and the possibilities to resolve it. For example, if some dominant problem stays unchanged for a long time, and if the factors which cause this problem are controllable, one can conclude that the developers' conceptions (concerning the information management strategy, the implementational strategy, project planning and scheduling, etc.) do not work effectively.

3. A SITUATIONAL APPROACH TO SOFTWARE DEVELOPMENT

3.1.Moving on to a more successful situation

The situation view of software development holds that the development is a function of the situation. This interpretation involves the following tenets:

- The software product results from a continuous process of feedback between the developers and the environment, e.g. the situation encountered.
- Each situation poses a decision making problem.

- On the situation's side of the feedback, marketing, technological and organisational factors cause changes in software development.
- On the developers' side, some psychological factors (motivations, stability, perceptions, attitudes, etc.) can influence on the decision making in software development.

We treat the decision making problem as a gap between the current situation and a more successful one. The latter reflects how we want the software project items look like. Thus, the decision making is focused on reducing the existing gap. Let the current situation be called IS and the more successful one - SHOULD. Both the situations are described on the same factors. To facilitate the decision making in software development, we modify the decision making procedure given in (Daneva and Maneva, 1993) by orienting it towards the software engineering problems:

Stage1. Define the IS and SHOULD situations in terms of all factors currently impacting on the software project. Establish factors' weights.

Stage2. Identify alternative ways of reaching the SHOULD-situation. Evaluate the alternatives on the base of the situational factors. The evaluations reflect the extent to which each alternative resolves the problem stated by the IS-situation

Stage3. Compare the alternative solutions and rank them according to its appropriateness to the IS-situation. The alternative with the highest rank is treated as the best solution.

Stage4. Select the solution after commitment. If the developers reject the best alternative proposed, they could adopt the next one. The selected alternative solution should be implemented.

Stage5. Control the implementation and evaluate the results against the SHOULD-situation formulated. If is needed, decide either to revise the SHOULD-situation, or to correct the solution implementation.

3.2.Modelling the software life cycle situationally

According to the situational interpretation, the software life cycle is modelled as a sequence of all situations encountered during the development process. The project has a partial life cycle during the time between any two consequent situations. The entire software development process can be traced by chaining the partial life cycles.

Each partial software life cycle involves the following phases:

Situation Analysis.

This means to determine IS-situation's PCR-s with respect to the relevant factor classes, to explore and to define the dominant problem fully so as everyone involved realizes its core. Finally, the SHOULD-situation has to be formulated.

Generation and Selection.

The decision group identifies the alternative solutions for problem resolving and selects the best one.

Implementation.

This means to plan an implementation procedure which establishes who should do what and when. The implementation has to be validated before the occurrence of the next IS-situation.

Validation.

The developers should devise an evaluation system so as to monitor the implementation of solutions and to determine the extent to which the implementation fits with the SHOULD-situation.

Documentation.

The results of each previous phase (i.e. situation's descriptions, alternative specifications and implementational solutions, etc.) should be carefully documented.

The crucial problem concerning situational approach is how to recognise the right time for occurrence of the next IS-situation. If two consequent situations are closely each to other, we can identify some unessential shifts. If checking is less often, one could miss some essential IS-situations. Hence, the developers have to strike a compromise between the benefits of detecting the situational factor changes as early as possible and the great efforts needed for identifying and analysing the IS-situations. Of course, there are some circumstances in which the necessity of considering the IS-situations is obvious. Such circumstances are: technological changes, market environmental turbulence, economic transitions, etc.

3.3.Developing the documentation

The quality of software documentation is defined in the terms of the content and the organisation of the documents [2]. The document base has to comprise information about both the final project of each phase of the software life cycle and the possible alternative solutions. Since our approach considers the situations subjected project development decision making, we propose to involve the specification of the situations in the final documentation. On the base of such a documentation the developers can derive decision rules about what can be done, if a certain situation occurs. These rules express the situation/decision structure of the project and can be used during software system maintenance.

All situations identified could be formally described by means of a proper specification language. The formal specification of the situations provides advantages concerning the following software engineering aspects:

1. Management reviews.

The situation documents could be treated as formal management reviews used in determining the overall progress of the software project and how the project's status conforms to the company's objectives. The analysis of the reviews may result in the management decision to stop software development or to change it.

2. Software benchmarking.

Describing and comparing situations is an essential step in the procedure for continuous improvement proposed in (Maneva et al, 1994). One of the main problems of applying it to software development is "to identify the situations at selected moments of the software life cycle which the procedure is used for". The thoughtful study of the environmental factors and the assessment of the situations results in identifying what is to be benchmarked, who are the comparative candidates, where is the gap, etc. Thus, the situation documentation provides a basis for controlling software development through benchmarking.

3. Version and release management.

According to Sommerville (1992), the crucial problem concerning system's versions and releases is "when the new release rather than a new version should be created". The careful analysis of the situations encountered during the creation of versions leads to determine the right time when the customers need a new release.

4. Decision making in re-engineering.

Restructuring and re-engineering decisions "can only be made on a case-by-case basis" (Sommerville, 1992). Hence, the situation documents become increasingly important. On the base of the situation documents the developers can identify the components which could benefit most from restructuring.

5. Software after sales service.

The situation documents resolve the problem of anticipating the users' needs and wants during the after sales service and support. Some alternative solutions concerning the free support, the daily and weekend support or the BBS/fax support could be derived. So the marketing staff is able to develop early the right service policy with respect to the software product considered.

3.4.Monitoring software maintenance

Usually, maintenance activities are associated with the final stage of the software life cycle, e.g. the maintenance begins when the product is accepted by the user and installed. From situational viewpoint we consider the maintenance as a process that accompanies the entire life cycle of the software system. We believe that the situational approach can involve the maintainer in the development process at every stage and provides him with an effective way of conducting periodic audit over the entire software life cycle. Having the situation documents, the maintainers can access all occurred problems and examine the impacts of the future modifications. Moreover, some known and described solutions can be reused in repeatable problem situations. The maintainers are able to concentrate themselves on the implementation of the new components instead of spending resources to access the history of the created system. Thus, the situational approach leads to decreasing of the maintenance efforts during production and after release.

4. PRACTICAL APPLICATION

We consider the situational approach as a powerful way of modelling the interactions between the environment and the software product development. In our opinion, its major practical application is to the development of projects under the following conditions:

a) the initial requirements to the software system can not be completely defined;

b) the development process seems to be highly dependent on the environmental factors;

c) the goals are unclear and there are many alternatives could be implemented;

d) it is necessary to model heterogeneous domain knowledge, complex entities and relationships and a distributed coordination;

e) the subject domain implies semistructured problems and repeatable situations subjected decision making.

Generally, these circumstances are related to the development of information systems handling both data and knowledge items. This implies the simultaneous exploitation of many interacting subsystems which may have their own independent models of operation. Next, there is a necessity to involve knowledge engineering practices in software engineering. To overcome the difficulties arising under the mentioned conditions, it is reasonable to use such implementational techniques that focuses on the closeness of the solution fit to the environment (Saxena, 1988). The interaction between the developers and the environment (the final user) can be provided in case of applying some prototyping implementational strategy. Prototyping facilitates the identification of some project snags, such as timing cycle bottlenecks of software incompatibility. The situational approach combined with prototyping encourage the project team to develop software in an open-ended fashion to minimize developers' tasks of facilitating repeated changes.

5.CONCLUSION

We have introduced a situational perspective to software development. Throughout the paper we have followed the idea to close the development process to the environmental changes. Thus, a situational approach to software development has been proposed. Our approach distinguishes itself from the existing evolutionary models (Bohem, 1988; Wild, 1992; Krasner, 1989) by the following features we consider as advantages:

1. It is an environment-oriented approach, which amalgamates eight factor classes into a generalized situation.
2. The software development process is managed on the base of situations encountered.
3. The progress on the project is continuously visible due to the situation documents developed.
4. The approach supports the decision making concerning both the software project and the final product. The described decision making procedure is applicable to resolve problems of setting prices, laying out a distribution network, making promotional plans , organising after sales service, etc.
5. The approach allows to anticipate not only the danger inherent in situations' changes, but also the new opportunities.
6. The approach implies the exploratory implementational strategies (prototyping, maquetting, etc.).

Special attention has been given to the development of the situations' documents. We have outlined how the software management and the maintenance could benefit from these documents.

Finally, the situational approach has been considered to be the most useful if applied to the development of intelligent information systems. The class of suitable software projects has been characterised.

This work is financed partially by the Ministry of Science and Education under the Contract I-24.

6. REFERENCES

Boehm, B.W., (1988), A spiral model for software development and enhancement, *Computer*, Vol. 25, pp. 290-302.

Wild, C., and Maly, K., (1992), Software life cycle support - decision based software development, Proceedings of the IFIP Computer Congress, Vol.1, Madrid, Spain.

Krasner, H., (1989), Requirements dynamics in large software projects. A perspective on new directions in the software engineering process, Proceedings of the 11th IFIP Comp. Congress, San Francisco, USA.

Sommerville, I., (1992), "Software Engineering", Addison-Wesley Pub. Co., New York.

Saxena, K.B.C., (1988), IS Prototyping, Hong Kong Polytechnic Technical Report, TR-8-03.

Earl, M.J., (1989) Management Strategy for Information Technology, Prentice Hall International Ltd., Hemel Hempstead.

Daneva, M. and Maneva, N., (1993), An intelligent marketing system, *IJ of ITA*, Vol.1, pp. 32-41.

Maneva, N., Daneva, M. and Petrova, V., (1994), Benchmarking in Software Development, IFIP Workshop on benchmarking, Trondheim, To appear.

DATA PROTOTYPING IS DEVELOPMENT

Andrej Kovačič, PRIS Consulting, 61000 Ljubljana, Slovenia

Abstract

The paper specifies a broad methodological view on perspective of Data Prototyping approach to the corporate IS development. Data Prototyping methodology would proceed in several different phases as follows: strategic information planning, information architecture development, conceptual and logical design, database and application software development, consolidation and environmental test, and quality control. Last phase evaluates efficiency and effectiveness of the corporate information infrastructure developed in former phases. Prototypes are conceptual modelling instruments. They relate to all phases of IS development. Data prototyping is in a context of conceptualization used as designer's vehicle for testing ideas, determining receptions, testing decisions and verifying quality of finally developed data base and application programs.

Introduction

Data Prototyping as an independent methodology for IS planning and development incorporates some fundamental principles of Data-Driven Prototyping approach. The methodology is a radical departure from traditional decomposition approaches to IS development and exploits advanced information technology. Data Prototyping uses abstraction approach focused on business processes and data entities in a system from which all knowledge of the business derives. On the other hand, data prototyping implementation methods focus on tools and procedures for database definition and transaction management. First they define the database aspects of the system, then they construct transactions that update the database. Finally, they develop transactions that retrieve data from the database.

Planning and development process, as seen from the data prototypers' point of view, can be divided into several stages, as follows:

- strategic IS planning,
- information architecture development,

- conceptual and logical design,
- database and application software development,
- consolidation and environmental test, and
- quality control.

The fundamental constructs of stages are exhibited in Figure 1. The standard data-flow technique is used to present different processes, data flow among processes, typical external entities and outputs or results of processes.

Strategic IS planning

The first task of management and IS specialists in a process of IS planning and development is to establish the Strategic Information System Plan which should be closely integrated into the business objectives of the organization (company). A plan which achieves the balance between the various demands made on the company resources by applying top down, bottom up and information weapon planning techniques.

Top down planning process begins by analysing the company's objectives, its strategies and critical success factors, and determining the information infrastructure required to support these. Bottom up looks at the current level of support of IS and determines the urgent and short term actions to solve usual problems. Information weapon planning refers to how a company may seek opportunities with which to use its IS to gain a competitive advantage.

Within Data Prototyping methodology, competitive strategy concepts developed by Porter are used. He views business as being pressed by five competitive forces: the threat of new entrants, the intensity of rivalry among existing companies, the pressures from substitute products, the bargaining power of buyers, and the bargaining power of suppliers. He also proposes three generic strategies to combat these forces: differentiation, which distinguishes company's products and services from others in all market segments, cost, where company attempts to become the low cost producer in all market segments, and focus or concentration of company on particular market segment and then differentiate or become the low cost producer in that segment.

Porter's concepts which are helpful in analyzing company's environment and business strategies, are developed by Wiseman to comprehensive framework, called the theory of strategic thrusts, for identifying IS opportunities. Strategic thrusts are major competitive moves made by the company to achieve advantage through: differentiation, cost,

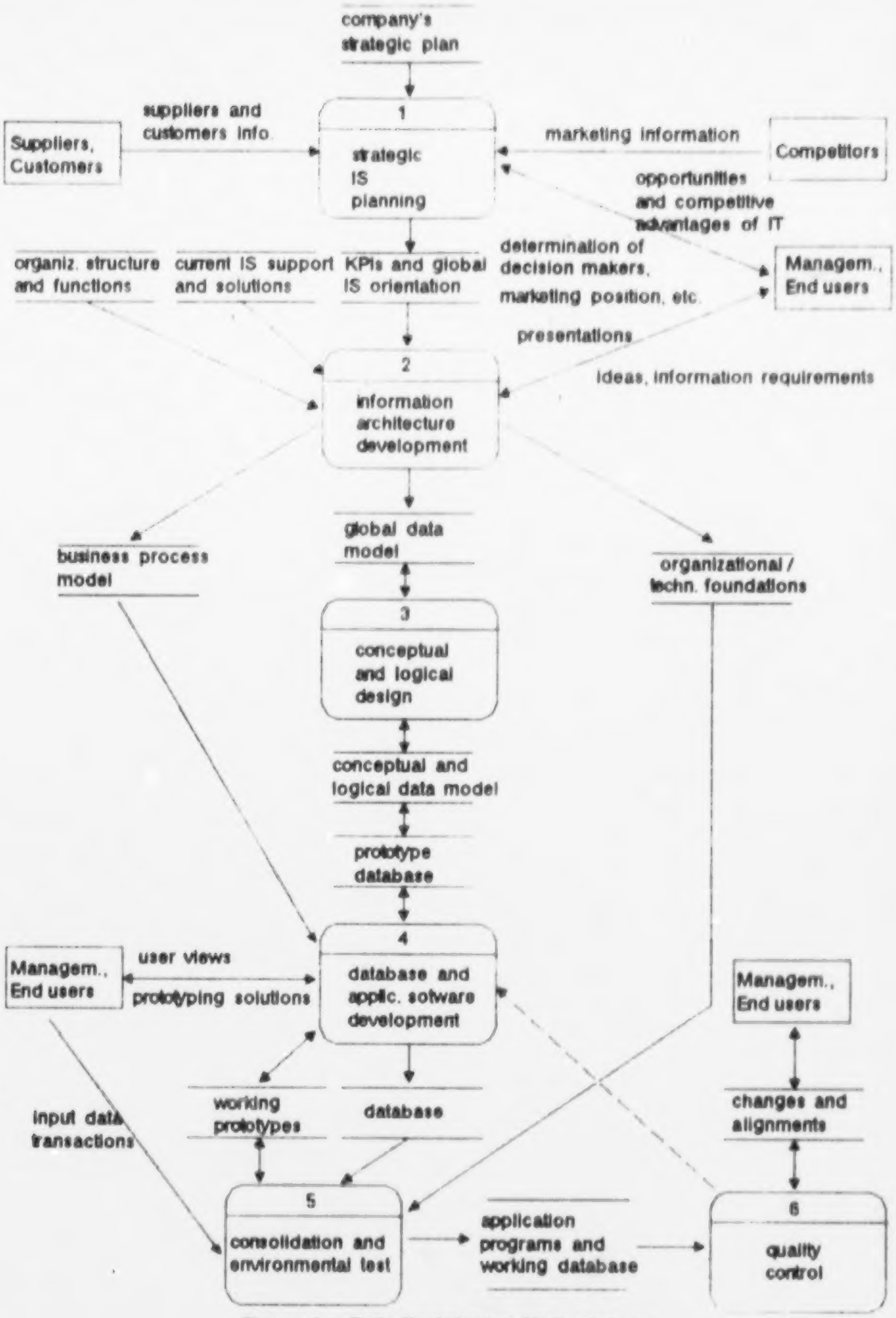

company's strategic plan
1
strategic IS planning
suppliers and customers info.
Suppliers, Customers
marketing information
Competitors
opportunities and competitive advantages of IT
determination of decision makers, marketing position, etc.
Managem., End users
organiz. structure and functions
current IS support and solutions
KPIs and global IS orientation
presentations
ideas, information requirements
2
information architecture development
business process model
global data model
organizational / techn. foundations
3
conceptual and logical design
conceptual and logical data model
prototype database
4
database and applic. sotware development
Managem., End users
user views
prototyping solutions
Managem., End users
working prototypes
database
input data transactions
changes and alignments
5
consolidation and environmental test
application programs and working database
6
quality control

Figure 1 : Data Prototyping Methodology

BEST COPY AVAILABLE

innovation, growth, and alliance. These thrusts strike the company at three classes of strategic targets: suppliers, customers, and competitors.

On the other hand, Data Prototyping is focused on reducing the amount of effort spent in a global top down planning process. This type of planning, based on exception principle of Pareto's Law, is termed an "80/20" approach. Using this approach we saw after the adage that for many undertakings, 80 percent of benefits can be achieved with 20 percent of total work. In this context another important approach toward the establishment of the possibility of using information as a competitive weapon is the critical success factors (CSF) technique.

Critical success factors are those few things that must go well to ensure success for a manager or an organization, and therefore, they represent those managerial or enterprise areas that must be given special and continuing attention to bring about high performance (Boynton and Zmud, 1989). In using CSF technique, which is derived from the same philosophy as the "80/20" rule, the first steps are to establish the objectives of the company as a whole, and determine its business strategy. The next step is to generate the critical success factors required to realise this strategy. This is done by electing critical information set from the top management and the key staff. The data obtained from the interviews and other sources is further refined and prioritised through group sessions during which the key performance indicators (KPIs) are agreed. Through KPIs such as stock and staff turnover, assets utilisation, waste factors the company can define higher return IS opportunities. To be able to calculate KPIs company must have access to appropriate data. These data are, in next stage of Data Prototyping methodology, used to develop the company's global data model which is a part of its broader information architecture.

Information architecture development

Information architecture is defined in this paper as the planning, designing and constructing information blueprint which can satisfy the information needs of business processes and decision making. It is derived from the KPIs and global IS orientations described in the previous chapter. Architecture calls for full recognition of the importance of data in the design and development of information systems, for a perspective which exhibits a balance between processes and data. Data planning must be a part of the early stages of information systems development. This is the central idea behind the data-driven approach to IS development (Appleton, 1983).

The crucial results of the information architecture development process are company's business process model (Process Architecture) , global data model (Data Architecture), and organizational/technological foundations.

Business process model or process architecture consists of a profile of major business functions performed in an organization, how they flow in a sequence, and the data which is transferred from one to the next. The major business functions are decomposed into detail processes. Our opinion is that, "data-flow" diagramming technique is most suitable to represent some of detailed design products of the process modelling. Process modelling is a necessary prerequisite to the data modelling, and needs to be iterative, with well defined deliverables. Here, and also in the further development of information architecture, rule "80/20" is used.

Determination of the global data model or data architecture is the next step in information architecture development. Global data model is presented as Entity-Relational model containing company's major data entities and business rules in between them. It reflects global information needs of the company.

Organizational/technological foundations or platform is referred to the computer hardware, software, communications network and programming tools by which computing and information resources run, are developed and delivered to users in a company. This platform addresses also the company's organizational question and the question: "How to organize IS resources to be best fitted to the company's business needs?".

Conceptual and logical design

In this stage of IS development we presume that company's global data model developed in previous stage contains its major information needs, and is a suitable foundation for further activities. Those activities are concerned into conceptual data modelling and logical database design. The final result of this stage is a prototype database developed for the particular selected application area or company's business function.

The conceptual design phase models and represents the user's view of data and, possibly, a specification of the processing or use of the information. The objective of this first phase is to produce a representation of the information requirements independent of the DBMS which will be used. Representation called conceptual design (schema) is expressed as an Entity-Relationship (E-R) model. The conceptual design procedure is divided into following steps (Storey, 1990), where we have to: Identify entity types and their attributes, Identify relationships and their attributes, Examine the design for potential problems and reiterate the first three steps until no more problems can be found or requirements demanded, and select primary keys from candidate keys.

During the logical design phase, a logical design (schema) that corresponds to the data model of chosen DBMS is produced. In the case, the logical data model is the relational model, the translation process involves the following steps: Representing and transforming the entities and relationships into a set of relations, and Applying the normalization principles to normalize relations.

Using CASE tools in the information architecture design stage and both phases of this stage, normalized relations of a logical design (schema) are directly transformed into physical schema (Tables in the relational database). Determining the physical parameters and database functional enhancement through constraints, triggers and stored procedures derived from a company's business rules, the prototype database of the treated business area or function, is automatically created.

Database and application software development

Prototyping is a design technique which recognizes that IS development is an iterative evolutionary process. It also recognizes that developers cannot always foresee and understand all sides to a problem and that the end users cannot always completely express their information needs.

One major constraint with which IS developers had to cope over the years is that the prototype became the production system (Dickson and Wetherbe, 1985). However, information technology is now flexible enough to make prototype or heuristic approach to IS development feasible. Recently, new interactive development tools have become available to enable design and evolutionary development of the database and related application software solutions. These development tools are generally based on fourth-generation language (4GL) integrated with a relational database and related with computer aided software engineering tools (CASE) or at least with data dictionary as its crucial part.

We believe that prototypes are conceptual modelling, design and implementation tools. They relate to all phases of IS development. Data prototyping is in a context of conceptualization used as designer's vehicle for testing ideas, determining receptions, testing decisions and verifying quality of finally developed data base and application programs. In the context of data prototyping a special emphasis must be put on the process of heuristic analysis and the role of end-users in this process. The final result of heuristic analysis is user knowledge and experiences captured in data base and application software (prototyping) solutions. Data prototyping as an iterative approach to database development practically eliminates the problems related to the integration of user views and enriches the quality of the finally developed database and application programs.

The diagram in Figure 2, adapted from Martin (Martin, 1982), represents the data prototyping procedures with special emphasis on the current stage of IS development. The left side of the diagram shows a top-down planning approach, described in previous stages, leading to the development of the global data model and prototype databases. In general, only selected database is chosen for further bottom-up heuristic analysis, design an implementation.

As soon as the selected prototype database is physically implemented, the prototyping effort starts. It starts with development of prototyping program solutions and their evaluation against existing and future information requirements scenarios. The prototype program solution is implemented and operated in a real context. The users hands-on operation of the prototype provides a basis for evaluation and learning. The designer solicits user feedback in terms of modification requests and new requirements. In response to this feedback, the prototype database is refined (if necessary), and programs are quickly modified. The users again operate and evaluate the revised prototype. The prototype evaluation and revision cycle is repeated as many times as necessary until a satisfactory prototype is obtained. In real, using appropriate prototyping tools, an average two or three iterative cycles are necessary to develop application solution satisfying current user's needs. Iterative development of prototype database is ended in the moment when all different inquiries to the database can be successfully obtained.

Consolidation and environmental test

During consolidation and environmental test stage of IS development, prototype databases are being put on the computer and logically and/or physically integrated. In direction of improving efficiency, the environmental test and tuning, which has responsibility for optimizing DBMS views and access, is obtained. Prototype application programs are improved from the efficiency perspective and can also be physically integrated.

In a database environment, the primary objective is to expose data to the widest audience of users (data sharing) while maintaining the integrity of the data and avoiding abuse of the database (data security). For those purposes, the database administration functions must be established. However, the data administration function must be established far more early, in the first stage of IS development.

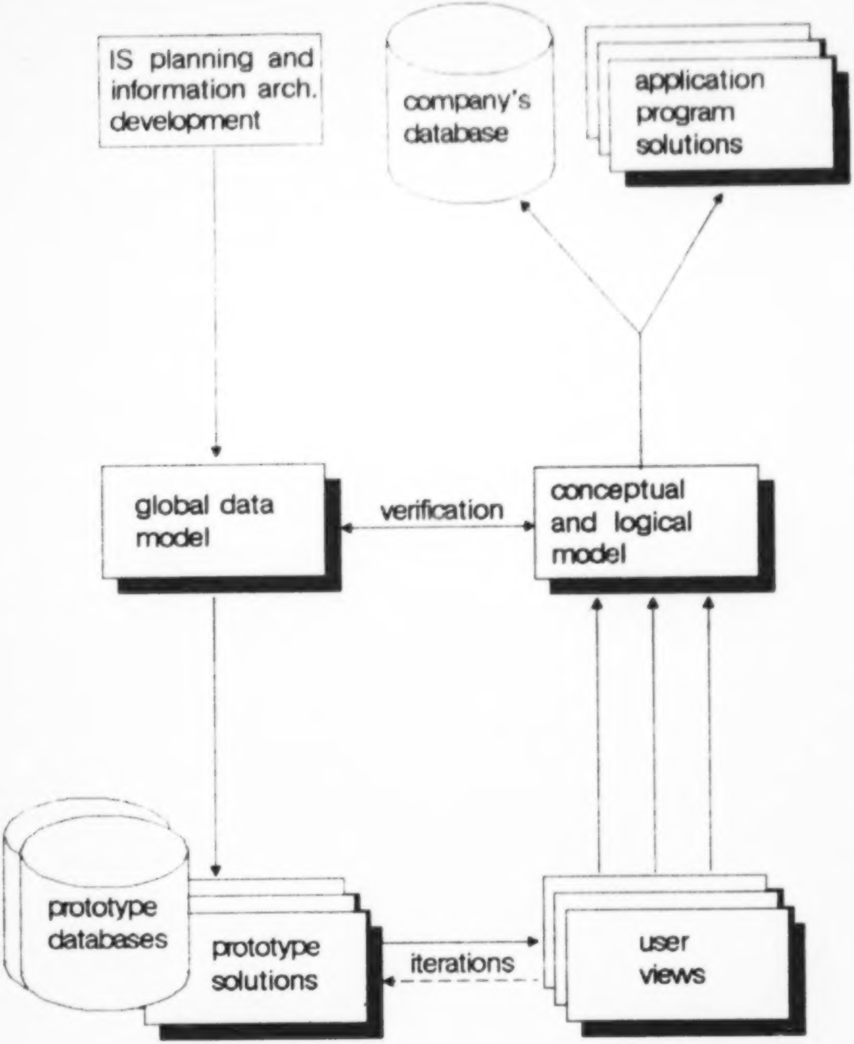

IS planning and information arch. development
company's database
application program solutions
global data model
verification
conceptual and logical model
prototype databases
prototype solutions
iterations
user views

Figure 2 : Data Prototyping Procedures

Quality control

During the quality control stage, which is the last but permanent stage of the IS development process, new requirements to IS are coming. The objective of this stage is to asses continuing alignments of the company's business needs and its information architecture. Requirements which change the information architecture must be followed by appropriate actions in all later stages of the IS development. Another major objective of this stage is a continuing integrity auditing of database and data dictionary contents.

Conclusion

The data prototyping methodology using advanced techniques, as heuristic development and prototyping are, represents a conceptual breakthrough in the area of information requirements analysis and database design and implementation. IS developed using this methodology and advanced techniques have greater effectiveness in determining and serving user requirements.

BIBLIOGRAPHY

Appleton D.S., Data-driven Prototyping, Datamation, November 1983

Boynton A.C., Zmud R.W., An Assessment of Critical Success Factors, in Shoof W. H. (Editor), The Management of Information Systems, Orlando, Florida: The Dryden Press, 1989

Brathwaite K. S., Relational Theory, San Diego, California: Academic Press, Inc., 1991

Dickson G.W., Wetherbe J.C., The Management of Information Systems, Singapore: McGraw-Hill, 1985

Martin J., Strategic Data Planning Methodologies, Englewood Cliffs, N.J.: Prentice-Hall, 1982

Storey V.C., Relational Database Design based on the E-R Model, Rochester, NY: University of Rochester, 1990

Wiseman C., Strategic Information Systems, Homewood, Illinois: Irwin, 1988

PROBLEMS IN INTRODUCTION OF THE RAPID APPLICATION DEVELOPMENT *(RAD)* PRINCIPLES PUT TO PRAXIS

Tone Ljubić[1] and Stane Štefančič[2]

[1]University of Maribor, School of Organizational Sciences, 64000 Kranj, Prešernova 11, Slovenia
[2]GENIS d.o.o., 61000 Ljubljana, Tržaška 40, Slovenia

Abstract

The Rapid Application Development *(RAD)* Method gives a possiblity of reducing time for developing computer application and at the same time to improve its quality by including end users into development process. When put to praxis, the implementation of ***RAD*** may cause certain problems, which have an important influence over the success of application development. These problems can be divided in various categories: general problems, problems in Requirements Planning Phase and in User Design Phase, in Construction Phase and in Cutover Phase of ***RAD*** and finally, problems in exploatation of applications. Main problems in particular categories are identified. Further on there are sources and causes of the problems explored and possibilities, how to solve and compensate these problems presented.

1. INTRODUCTION

1.1 STARTING POINTS

Rapid Application Development *(RAD)* method as elaborated and proposed by James Martin, is a generalized life cycle of a software applications development, designed to construct applications much faster and of higher quality than the traditional life cycle. ***RAD*** offers significant savings in the development time and an improving of system quality.

However, the ***RAD*** demands radical changes in the way people who are involved in applications development think and work, including the management. We find great success in the environment, which accepts and performes such changes. But where the climate is not favourable to the changes, the success often fails. The reasons for such failures are rather banal than serious and can be eliminated relatively easy.

1.2 CHARACTERISTICS OF *RAD*

Main characteristics of ***RAD*** are:

- Three-phase life cycle with duration of approximately three months for smaller applications and four-phase life cycle with duration of approximately six months for bigger applications, these phases are
    - Requirements Planning Phase
    - User Design Phase (in a three-phase life cycle these two phases are united)
    - Construction Phase and
    - Cutover Phase

  as shown in Figure 1.

- Intensive cooperation between end users (workers or employees performing bussines processes) and IS professionals (system analysts, programmers, data base designers etc.). Complete work in initial phases is done by mixed teams - Joint Requirements Planning (JRP) Team and Joint Application Development (JAD) Team, which perform their work at workshops. Only in construction phase all is done by IS professionals - but the results are checked by end users. In the cutover phase a cooperation of end users and IS professionals is requested again.

- Use of highly specialized integrated CASE tools (I-CASE) for creating system specification and automated coding of computer programs with appropriate data base designers and application generators.

1.3 ENVIRONMENT IN WHICH *RAD* WAS APPLIED

After being anounced, the ***RAD*** was applied in one Slovenian enterprise, engaged with information processing services with outsourced data processing and applications development for clients. ***RAD*** principles were tested in two very different environments, in banking and in chemical industry. The JRP in JAD teams were mixed with members from the both sides, the teams for construction of applications were formed only by IS professionals.

There was no great success in the beginning. People must get some practice and experience in the first place. The problems of this period are discussed in the continuation of this article. After the troubles in this starting period ***RAD*** becomes a usual way of work, the results are appropriate.

2. PROBLEMS IDENTIFIED

2.1 GENERAL PROBLEMS

2.1.1 The enterprise identity and strategy are not identified

A modern enterprise should be aware of its own purpose, philosophy, objectives and goals of its bussiness processes and functions. The strategy, how to achive these goals, also must be known.

- If the management is not able to identify objectives and strategy of the enteprise, it is an illusion to expect, it will give appropriate priorities to use time, resources and money for construction and develoment of organization and information system.

  To do this, the management should
    - ensure the uniformity and sinergic effects of the enterprise as an entirety
    - rationalize the performing of important common tasks

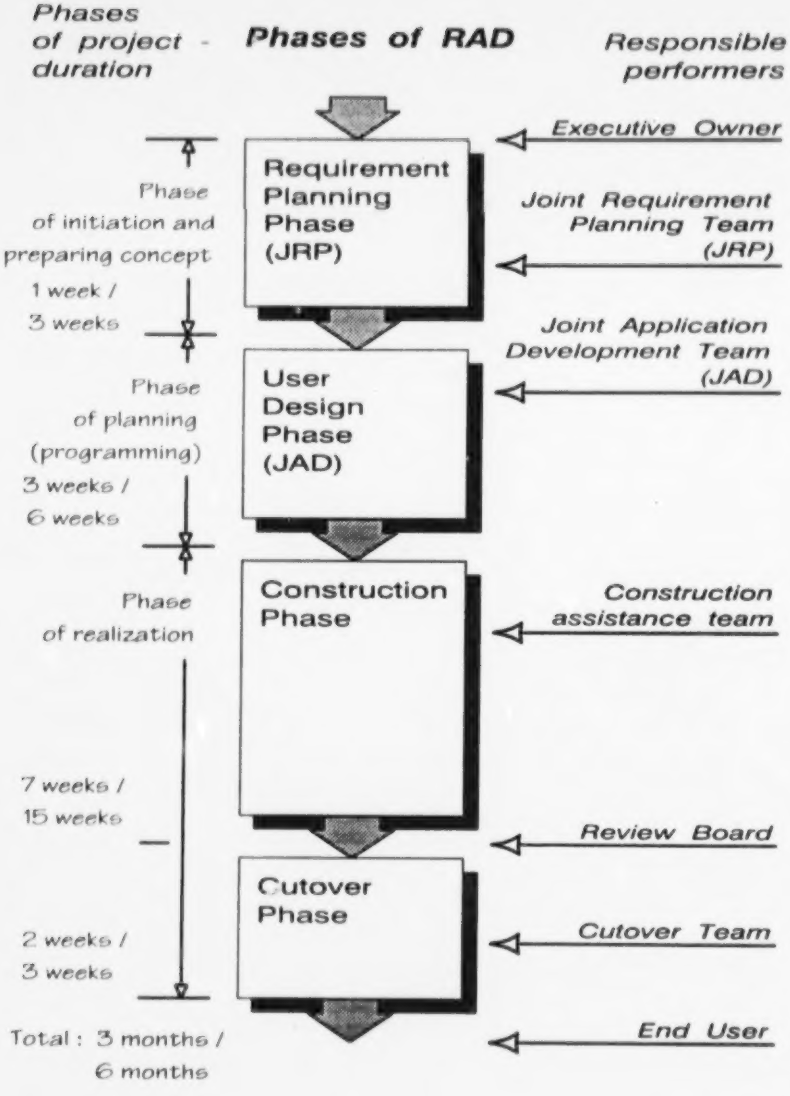

Phases of project - duration
Phases of RAD
Responsible performers
Executive Owner
Requirement Planning Phase (JRP)
Phase of initiation and preparing concept
1 week / 3 weeks
Joint Requirement Planning Team (JRP)
Joint Application Development Team (JAD)
User Design Phase (JAD)
Phase of planning (programming)
3 weeks / 6 weeks
Construction Phase
Phase of realization
Construction assistance team
7 weeks / 15 weeks
Review Board
Cutover Phase
Cutover Team
2 weeks / 3 weeks
Total : 3 months / 6 months
End User

Figure 1: The principles of Rapid Application Development (***RAD***)

- round and integrate bussiness processes into appropriate organization units
- emphasize the functions, which will be of a great importance in the future, when the hardest conditions are expected
- assure exactly defined limits of responsibilities and competences
- secure, that the tasks of vital importance for the enterprise are identified and very high priority is given to its performing

2.1.2 Management doesn't recognize real importance of information system

In many cases management considers the information system of the enterprise to be a stand-alone system, which has got nothing in common with real life. Information system is equalized with computer equipment and computing and is not treated as an important tool for gathering, processing and distributing information. Their opinion is, that they are able to manage the bussines on the basis of their intuition, they do not trust the information.

- In such case management gives very low priority to all tasks concerning developing and running information system. In most cases money for the computer equipment is provided - because a nice monitor on the desk of a manager is considered to be a status symbol - but the computers run in vain, without making information impact on bussiness.

  The information system neither can be selfsatisfactory nor can be a reason for its own existence, but it supports the organization system of the enterprise and is tightly bound up with them. It is impossible to make business without relevant informations. Therefore the investments in information system are of equal importance as investments in production capacities, marketing etc. But - even the most modern computer equpment is not an information system!

2.1.3 The meaning, information system shall last for ever

There are still applications existing which were developed thirty years ago for the punched card equipment, later transfered to IBM /360 or similar systems with classic data files and batch processes, then adapted to use early data base management systems, again modified to work in interactive environment and finally downsized to fit the personal computer platform - but with the initial logic retained. Generations of end users were using these applications, they have these applications in blood and bones.

- The same end users will ask: Is it really necessary to throw these (from their opinion well behaving) applications away and to develop something new? Is there nothing more reasonable to do?

  It must be clear to the top management that the information system and applications supporting it are not everlasting, they have a limited life time. The reasons for this are changes in bussines functions and processes, which are currently not supported by obsolete applications, as the capabilities of newer information technology which allows the processes to be performed faster, more reliably and closer to the end user.

2.2 PROBLEMS IN REQUIREMENTS PLANNING PHASE AND USER DESIGN PHASE

2.2.1 End users are not able to identify and define their needs

In the phase of requirement planning the JRP - Joint Requirement Planning technique is used. In JRP workshops, which are conducted and guided by IS professionals, the members of middle management (management of the business function for which the IS should be developed) and key end users must identify and define their needs.

The troubles which appear at this step are mainly:

- End users participating in the JRP workshop are not familiar with all the processes of the business function which are treated. So the information passed to the developers is not complete and correct.

  The only remedy is enlarging the circle of end users with the performers of specific (sub)processes, which usually are not the members of JRP team. They can be adopted to the team for the whole time of work or only for the time they are needed.

- End users are not able to estimate the volume of data and information they realy need. Usually the needs are overestimated, including non relevant data with no exact source defined and information which are unnecessary.

  In such case the skills and professional knowledge of a team leader (which is normaly IS professional) are very important. He must guide the participants through the problem and insist on clearing all details.

- End users are rather conservative than progressive. They like to keep old, traditional, "reliable" procedures although these are clumsy and not rational and are not very happy to exchange them for modern, "not tested enough" methods.

  To avoid this phenomenon a continuous education of end users is required. The education should be wide, not related tightly to the professional field of performers of concrete processes, with respect to wider environment of these processes - especially with respect to information processing.

2.2.2 End users can not communicate with IS professionals

As mentioned, JRP workshops and especially later JAD - Joint Application Development workshops are guided by IS professionals, in the phase of user design also other IS professionals must participate actively.

The communication between end users and IS professionals is very difficult in many cases for various reasons:

- The IS professional, who guides and moderates the JRP workshop, has very few or no knowledge about the processed area, so he is not able to direct the discussion in the right direction. The results are deficiently defined needs with uncleared details.

  An IS professional for guiding the JRP must be chosen reasonably. For end users a continuous education of IS professionals is of vital importance.

- IS professionals are looking down on end users, considering them to know nothing about information systems and information technology and not allowing them to "interfere" their solutions - though these are not optimal from the end user's view. End users are ashamed because of this and they are - directly or secretly - no more ready to work in JRP/JAD team.

  It is important for all participants in ***RAD*** to recognize the fact, all of them are equally important in achieving the goal setted - an optimal, useful and from the side of end users accepted application. People, who are acting with arrogance, who consider themselves to be better than others, should have no place in ***RAD*** teams.

2.2.3 End users are not ready to join JRP/JAD teams

- Especially older people, who otherwise are experts in their professional field (but are in many cases formally undereducated), do not like to work together with young peoples, because they fear to be outloughed.

  The solution of this problem is the same as the solution of the problem of deficient commnunication of end users with IS professionals, mentioned in the previous paragraph: all members of the teams are equally important; the experience is worth as much as the knowledge.

2.2.4 Lack of time to work on application development for the members of JRP/JAD teams

Concept of ***RAD*** requires "full time job". Everybody, who is included in various project teams and is invited to attend the meetings of the teams, must work full time on the ***RAD*** project. When his task is fullfiled, he is free to other jobs.

- Some people don't find it easy to quit their usual work and to join a ***RAD*** project team for a few days. They think, their "regular" work will suffer and the loss will be greater than the benefits of applications being developed. They search for excuses and suggest a partial participation at meetings (maybe half of a day, every second day etc.) or being replaced by other people. The work on application development suffers, the needed work is not done (deficient results) or is done by other participants, which are not in the swing of things on the whole.

  This is the question of the enterprise strategy. Nobody is irreplaceable. If information system is supposed to be of a vital interest for the entreprise, such deviations are simply cleared with a decree, which nominates another person to substitute the one who is absent and will work on application development for the period of his absence.

2.2.5 IS developers are not familiar with tools used for design

Normaly in a JAD phase the use of I-CASE tools for designing the application begins. I-CASE tools are very specialized and sophisticated tools, many times not transparent and not very friendly to user; these tools require a solid background in mastering the principles of information engineering and technology.

The problem of communicating end users and IS professionals is here repeated. Beside of this additional problems arise:

- The IS professionals do not master the tool used. They are not familiar with all capabilities of the tool, with procedures to perform something or with results presented. When end users, participating JAD workshop, see this, they lose the confidence in the developers and in the developed product.

  The only medicine: education and training.

- Some IS professionals have their own private favorite tools, which are not compatible with officially used tools. Redundancy is inevitable; several processes of development are performed twice, in most cases with different results.

  In one ***RAD*** life cycle only one tool should be used. The use of this tool should be

247

demanded strictly, no other tools should be allowed. Introducing a new tool realy must be justifiable (new, extended features etc.)

2.3 PROBLEMS IN CONSTRUCTION PHASE

2.3.1 Lack of appropriate I-CASE tools

RAD is meant for the use of an integrated CASE tool through all phases of a life cycle.

- I-CASE tools available in many cases are not fully integrated. Their weak points are either at the upper end (treatment of bussines processes) or at the connections to data base design and application generators.

  The CASE technology developes permanently. IS professionals are obliged to follow novelties on the CASE tools market and to change from time to time obsolete tools with newer, more capable solutions.

2.3.1 User's specifications are not considered

In the construction phase the application is technically developed by IS professionals: phisycal design of the data base is done, the transaction programs are coded. All this is done on the basis of end users' requirements and definitions, done in the previous phases.

- To the opinion of IS professionals involved in the construction of application the end users absolutely are not acquainted with the problems of application development so their specifications are worthless. IS professionals make their own specification with no consultation with users, adapted to the technology available and not to the needs of end users. These specifications are used for the construction of the application rather than the end users' specifications.

  It is important that the users' requirements and specification do not change without previous consultation and agreement of the end users. If the requests can not be carried out due to technical problems, the steps of requirements planning and user design should be repeated.

2.3.2 Missing integration of different IS modules

Big systems handled by ***RAD*** should be divided in smaller modules, small enough to be developed in a three or at least in six months life cycle.

- The development of different modules is performed by different teams. Because of specifics of the teams (different tools, different way of work, different time of work, deficient communication between teams) the results of development are not fully compatible and can not be integrated in a comprehensive information system. In most cases this is expressed as redundance in data bases or in a different appearance of screens, reports etc.

  Development of big systems must be treated as multiproject with development of smaller modules as subprojects, managed and coordinated by a multiproject management team. Especially the data bases should be designed in common with a level of multiproject their design and contents becoming obligatory to all development teams. The standardization of the components of the applications should be carried out (standardized meaning of function keys, appearance of screens and reports, help etc. for example).

2.4 PROBLEMS IN CUTOVER PHASE

2.4.1 Insufficient final testing and audit

In the cutover phase normaly quality audit of the developed application is performed and final, logical testing is carried out. After auditing and testing the application should be error free and certified for the use.

- At the end of ***RAD*** life cycle time becomes short. The developers are expected to work well and to be dilligent, making no mistakes, therefore the quality audit is very superficial or even none. The testing also is done in a short time, with test data adapted to desired results. The testing with real, "live" data in real environment is not performed, assuming the application has a period of warranty in which the defects are to be repaired.

  The ***RAD*** project manager has to be persistent and insist on performing quality audit and completing procedures of testing even if the delivery term for the application will be excessed.

2.4.2 Impossible parallel running of the existing and new system

Parallel running of the existing system and new application which should replace the previuos one is considered to be absolutely the best way of testing the application. Both systems must run at the same time with the same data, results should be identical.

- In the same cases it is objectively technically and organizationally impossible to run two systems parallel. The meaning of cutover is literal in this case; it means "to cut", old system is withdrawn and the new system becomes fully operational within a moment.

  This is possible only if the new application was tested intensively and is realy error free. Despite of this it is recomended to do such transition in the otherwise non-working days (during the weekends or holidays). Transition should be planned exactly; the best way is to create a network plan (PERT chart), where the activities and tasks to be performed are specified, the terms (day and hour) of these are defined as resources and responsible people.

2.4.3 End users don't get what they expected

- When the application is put into final testing by end users, they are surprised and disappointed because the outlook and/or results of application are not what they expected.

  This indicates either the specifications in a requirement planning and use design phases were formed superficialy regardless of all details or the end users' specifications were disregarded in the construction phase. Unfortunately very little can be done in such case. The application might be corrected, end users' requests might be reduced and adapted to the actual possiblities of the application. But the consequences must be drawn - such case should repeat never again.

2.5 PROBLEMS IN EXPLOATATION

2.5.1 Wishes and needs for changes arise very soon

Well defined, designed and constructed applications should be in a stable environment in use for a relatively long time without radical changes.

- After a short time of the use of application the end users are not satisfied with it. They suggest it should be changed

The reasons are practicaly the same as stated in the previous paragraph requirements were not clearly disposed and specification were not precise, or end users' specifications were disregarded at the construction In many cases end users do not express objections at testing and cutover fearing to be regarded as a drag in progress, they accept application as it is - even if it doesn't correspond to their needs and wishes

Such application should be corrected as soon as possible

3. CONCLUSION

Most of the mentioned problems have already been pointed out by James Martin Our experiences only may confirm his presumptions

The success of applications, developed with ***RAD*** technique, depends not only on capabilities of IS professionals involved in the development Knowledge of end users is equally or even more important, their ability to specify their needs with respect of possibilities of information technology being used and their willingness to use these applications

A high level and a wide professional knowledge and skill are requested in ***RAD*** It is absolutely necessary to assure a continuous education of all people involved in any development activities (this is true of all developments - not only of IS applications development!) The areas of education should be mixed, so for example technical experts should be educated in informatics and vice versa IS professionals should be acquitained with problems of bussines processes in an enterprise

RAD requires team work People in most cases do not know how to act and how to behave in a team, team leaders do not know how to lead teams, how to moderate the discussions, how to stipulate results and how to stimulate the members So the education in specifical managerial techniques (Human Resource Management) is needed too

The possibility of the initial failure of ***RAD*** should not discourage people to proceed using this method We all learn by making mistakes - next time the things will run better

4. REFERENCES

Hollocker, C P , (1990), "Software Reviews and Audits Handbook", John Willey & Sons, Inc , New York

Martin, J , (1990), "Rapid Application Development", The James Martin Productivity Series, James Martin Report, Inc , Marblehead, MA, USA

Page-Jones, M , (1985), "Practical Project Management, Restoring Quality to DP Projects and Systems", Dorset House Publishing Co , Inc , NewYork, NY, USA

Rant, M., Jeraj, M., and Ljubič, T., (1992), Enoten kompleksen sistem planiranja v proizvodnih podjetjih *(Unified Complex System of Planning in Manufacturing Enterprises)*, POIS, Radovljica

TOWARDS CONTINGENT INFORMATION SYSTEMS DEVELOPMENT APPROACHES

Kees van Slooten[1] and Bram Schoonhoven[2]

[1]University of Twente, School of Management Studies, P.O. Box 217, 7500 AE Enschede, The Netherlands
[2]RCC Information Services, P.O. Box 9105, 7300 HN Apeldoorn, The Netherlands

Abstract

Situational methods based on project contingencies become more and more an important research topic for information systems development organizations. A project characterization, based on a matching process between dominant contingency factors and the pre-conditions of possible approaches, before and during the information systems development process is necessary, in order to accommodate the specific circumstances and requirements of a project. The information systems development department of an information services organization is the environment of the field study, whereof some results are exposed. A meta process model of the information systems development process may facilitate the generation of situation-specific information systems development processes, but some basic information systems development approaches have emerged. Contingency-based conditions can be added to the basic or constructed approaches.

1. INTRODUCTION

In practice, the linear way of working during information systems development is abandoned due to specific requirements of the specific situation. Different circumstances, due to different application domains, interest groups, business strategies, cultures and skills, require a different approach, a different collection of methods and tools, and the performance of a different set of development tasks in a different sequence. Van Slooten et al. (1994) define contingency factors as follows:

> Contingency factors are circumstances of the project influencing in some way the selection or construction of an approach (situational method) to systems development.

Situational method engineering has been defined by Van Slooten and Brinkkemper (1993) as follows:

> The process of configuring a project scenario for information systems development using existing methods, or fragments thereof, to satisfy the factors that define the project context.

A scenario is an approach to the systems development process or, in other words a situational method, determined by contingency factors. The configuration procedure proposed by Van Slooten and Brinkkemper (1993) consists of the following stages:

- Characterization of the project by determining the important contingency factors and by deriving from these factors so-called intermediate variables, namely: development strategy, ways of modelling (aspects), levels of detail and situational constraints. Van Slooten et al. (1994) have mentioned two additional intermediate variables: the roles of the participants, and the relationship between the project and its environment.
- Composition of a first project approach by the selection of the most appropriate method fragments and the route map. Route maps are development strategy plans consisting of activities and products. The composition is supported by a computer aided method engineering tool.
- Further refinement of the approach during the course of the project. The complete performance of the project leads to more development expertise, which is exploited by future projects.

The *focus* of this paper is on the determination of the development strategy and under which conditions a particular development strategy is feasible, which means that the use of the term "approach" must be interpreted in this restricted way. We concentrate ourselves on the relationship between project characterization and development strategy.

The empirical research, a kind of field study, has been accomplished in cooperation with the staff of a Dutch information services organization in the domain of information systems development: RCC. In the past, RCC has developed computer-based information systems on a mainframe hardware platform, nowadays also on other platforms. The development of large-scale systems is supported by SDM, which is the adopted Systems Development Method. SDM is a widely used methodology in the Netherlands and follows a typical linear development strategy. RCC has indicated the following trends in information systems development:

- Increasing dynamics in the environment of the customer organization, which means that information must be more flexible and changeable due to changing requirements and circumstances.
- Increasing deconcentration and decentralization of the information service function, which means more diffusion in the organization of the development and use of information systems and more responsibility and decision making for the user.
- More automation also means a shift of knowledge about automation to the user organization with as a consequence more criticizing from the side of the user.
- The evolution of technology (hard- and software) is an important incentive for down- and right-sizing as well as approaches like prototyping.

Since RCC also develops systems on other platforms, there is a need for the development of more small-scale and flexible information systems. More decentralization and technological development necessitates the application of other approaches to systems development like prototyping and incremental development. New approaches to systems development cause a different way of working. Although the importance of new approaches to systems development has been recognized within RCC, the practice has not been changed yet sufficiently. However, nowadays more alternative approaches besides the SDM approach are applied, which is caused by the following developments:

- Increasing use of CASE-tools makes it possible, necessary or desirable to apply another approach.
- Increasing need for support of project teams using a prototyping strategy.
- RCC wishes to acquire more small-scale projects like client-server, PC and PC-LAN applications in the future.

This means that there is a need for more situation-specific project approaches, i.e. an approach accommodating the project context to support the project team. Furthermore, it is important for RCC to gain more experience with other approaches to the systems development process. A topic of our research is the development of tools enabling the project manager to select a proper approach. A model for the choice of an approach is elaborated in section 2 of this paper. A meta process model for the generation of situation-specific development strategies is discussed in section 3. The basic approaches, described in section 4, are based on this meta model for the primary process of systems development. These basic approaches are selected by comparing the project-specific contingency factors with a number of pre-conditions specific for a certain basic approach. Conclusions and further research can be found in the last section. Schoonhoven (1993) contains more detailed information about the field study.

2. THE DETERMINATION OF AN APPROACH

2.1 The Need for Contingency-Based Approaches

The linear approach, based on the waterfall model, is appropriate for the development of information systems when the specifications are stable and clear. However, nowadays new application domains are less structured and specifications become less obvious and are often subject to change. There are many other disadvantages related to the linear approach, e.g. the emphasis on phase products instead of the end product; organizational changes during the development process, which are often not considered. These and other factors have led to alternative development models, e.g. incremental development, evolutionary development, rapid prototyping, the spiral model (Boehm 1986). Those non-linear process models are supported by new technological development, e.g. CASE-tools and fourth generation languages. Through the emergence of different process models (approaches) and the situation-specific nature of each model, it is necessary to characterize the project, which enables the selection of the right approach.

The first contingency-based approaches have been developed by Naumann et al. (1980), Davis (1982), and Burns and Dennis (1985). Baskerville et al. (1992) proceed in this direction and state that the selection of an approach, based on fixed criteria is not sufficient. Furthermore, they argue that an approach to information systems development must be developed, just as an information system. In this way, approaches are emergent, which means that they have a short lifetime.

In practice, the relationship between situation and project approach is bi-directional (Van Slooten et al. 1994), which means that the situation influences the project approach and vice versa. A contingency-based, situation-specific project characterization occurs preceding systems development (ex ante), but also it may occur and it usually occurs during project performance (on-the-fly). During the initial characterization one might choose between two alternatives. On one side, one might choose the most appropriate approach from a set of available approaches (the selection alternative); on the other side, one might construct a completely or partially new approach accommodating the specific situation (the construction alternative).

2.2 A Process of Approach Determination

An approach will be considered as a coherent set of rules and characteristics indicating the way a systems development project is carried out. The focus of the characteristics is on the goal, the product and the process of the information systems development project. An approach will be considered as a global strategy and not as a cook-book. Ould (1990) already represented development strategies through process-based models of the systems life-cycle, but this was not based on a generic model. In this paper, we also choose the process as the basis for the classification of information systems development projects. The availability of a number of possible project approaches is very useful for the practice of project management. Those classes of project approaches may contain, for instance, a specification of phases, the sequence of phases, a way of user participation and possible pitfalls. Boehm (1989) has formulated this as follows: "What we really need are process model generators", which is exactly the subject of this paper and is supported by our process of approach determination including our meta process model. We wish to describe a process for the determination of a project approach with the possibility to learn from experience.

We have already mentioned a few contingency-based approaches for the determination of a project approach. Furthermore, we wish to say a little bit more about two topics: risk analysis and the determination of critical success factors.

- Risk Analysis

Often one risk number is derived from a number of different contingency factors. A well-known example of such a calculation is published by Naumann et al. (1980). Naumann et al. propose four groups of contingency factors, from which a development strategy is determined through the calculation of a level of uncertainty. A low uncertainty number corresponds to an "accept user statement" strategy, a high uncertainty number corresponds to an experimental development strategy and a number in between corresponds to either a linear or an iterative development strategy. However, in practice, different risks might be solved in different ways. Sometimes an organizational solution is more appropriate. Furthermore, Naumann et al. made the implicit assumption that development strategy choices can be made independently of the source of the uncertainty. Moreover, the subjectivity of such calculations is a real danger. The conclusion is that such a risk analysis is only useful during a project characterization to obtain an initial, global idea about a possible development strategy, but a more specific project characterization is still necessary using specific knowledge from experience.

- Critical Success Factors

Critical success factors are entities, e.g. activities or conditions, which are crucial for achieving the project goal. After the determination of potential critical success factors, measures are taken in order to prevent problems. Critical success factors may play the role of contingency factors during the project characterization. Van Slooten et al. (1994) emphasize the success or failure behaviour of contingency factors and mention a few examples, e.g. team spirit, involvement of all participants, early and clear decision making.

The mentioned contingency approaches are at most partial solutions for the determination of a development strategy as a component of situation-specific information systems development. Van Slooten et al. (1994) state that we have to deal with a number of aspects of contingency factors during project characterization, namely: duration, direction, scope, deepness, origin, and mutual relationships. They also mention some problems with the determination of contingency factors: the unit of the factor, the visibility of the factor, the measurability of the factor and the stability of the factor. Also Lyytinen (1987) mentions a number of existing problems of contingency approaches to systems development:

- The current contingency approaches are only partial solutions.
- The emphasis of existing contingency approaches is on measurable situation factors and use a strictly cause-effect interpretation scheme, which means that the cultural context is neglected.
- The proposed frameworks have an ad-hoc nature.
- Ambiguity arises if one tries to apply the models to specific development processes.

Figure 1 presents a model for the determination process of approaches to systems development. This is used by the remainder of this paper and is actually a partial elaboration of the configuration procedure proposed by Van Slooten and Brinkkemper (1993) restricted to the determination of a development strategy.

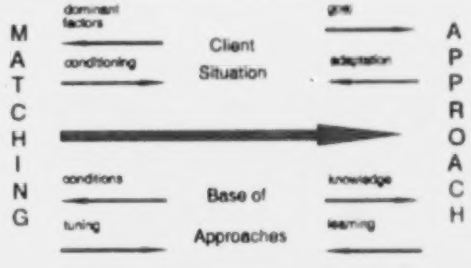


Figure 1. Approach Determination.

On one side, an approach might be determined based on dominant factors in the client situation (figure 1), on the other side based on pre-conditions that must be satisfied when applying a related possible standard approach (base of approaches in figure 1). The standard approaches have already been proved useful in practice under the related conditions. In this way, an organization is able to build its own knowledge base or method base (Van Slooten and Brinkkemper 1993) with possible approaches, that is consulted in similar situations. To make a proper decision about an approach, there must exist a match between the dominant factors of that particular situation and the pre-conditions of the approach. A possible approach might be tuned to fit the actual situation better, or sometimes the situation is conditioned with the intention to facilitate the application of a particular approach (Van Slooten et al. 1994). In fact, we distinguish two options for the choice of a specific approach. One use of a reasoning mechanism to bridge the gap between the contingency factors and a possible approach. The other option is the check of the pre-conditions of possible approaches for that particular situation. We think that we have to implement an integration of the two options. Both are necessary for the determination of a specific approach to information systems development.

3. THE META PROCESS MODEL.

Many problems with information systems development projects arise from mismatches between the process model (e.g. waterfall, spiral model, prototyping) and project contingengies (e.g. budget, technology, customer standards, development expertise and time). The primary process modelling approach to date, for avoiding these mismatches is trying to develop one "best" process model, that works well for any combination of project contingencies. We propose to use a process model generator, our meta process model, making it possible to construct situation-specific primary process models. Our meta process model consists of two cycles of the primary process: the development cycle and the operations cycle (figure 2). The addition of the operations cycle makes it easier to describe completely the possible approaches, which is illustrated in the remainder of the paper. Around the primary process of systems development, four different management processes can be arranged: project management, user management, information systems planning and quality assurance. The management processes may play an important role within the primary process depending on the nature of the process.

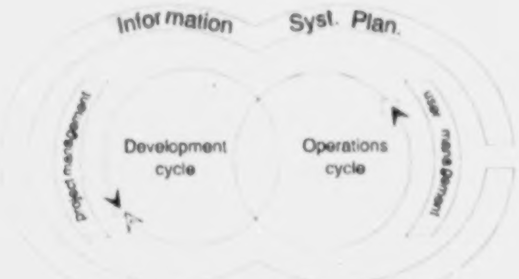


Figure 2. The Meta Process Model.

Figure 3 contains a possible instantiation of the primary process, which is greatly self explanatory. The terms "object systems analysis and design" and "information systems analysis and design" have been precisely defined by Van Slooten and Brinkkemper (1993). Object systems analysis and design aims to design a new object system by articulating and solving the problems of the old object system. Information systems analysis and design aims to design a computer-based information system through the analysis of that particular part of the object system, that has been selected by object systems analysis and design for that purpose.

Each approach to information systems development is an instantiation of the meta process model, e.g. different routes through the cycles with different goal/product combinations to realize, different actors and different method fragments used by the processes. The different components of the information systems development process are described. Using these components one can construct a specific process model: the route through the processes.

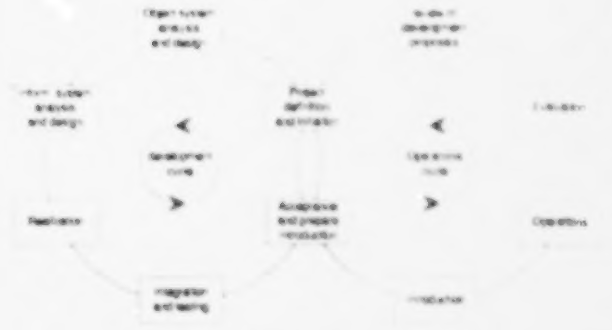

Object system analysis and design
Information system analysis and design
Project definition and initiation
Evaluation
Development route
Operations route
Realization
Operations
Integration and testing
Introduction

Figure 3. A Primary Process Model.

One can start a project from three possible starting points: Information Systems Planning (e.g. new system), Operations (e.g. maintenance) and Development (e.g. prototyping, incremental). When a project is initiated a process route is chosen through which a certain product must be delivered and a goal must be achieved. In figure 4 the different components of a complete process are given. Figure 4a shows the development of a subsystem or prototype, the results are demonstrated for future users, which means that the developed product is not used immediately, and the decision must be made what must be developed during the next development cycle. Figure 4b shows the development of a "throw away after use" system, which means that the system will not come back to the development cycle for maintenance. Figure 4c shows incremental development. After the development of a part of the desired information system, this part will actually be used and another part will simultaneously be developed. Figure 4d shows the development of a (part of a) system, that after it has been in operation for a while, returns to the development cycle, e.g. maintenance or slowly growing systems. Figures 4e and 4f show the reverse engineering principle.

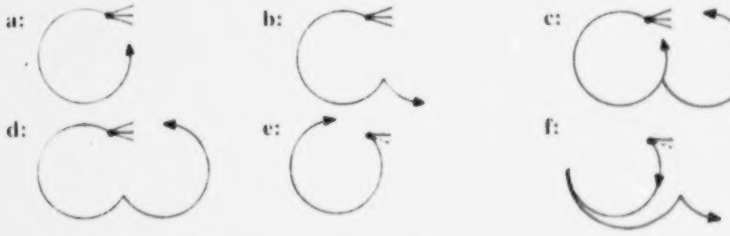


Figure 4. Components of the Primary Process

The different components of the information systems development process are used to compose a project-specific process model. A model of the information systems development process is only one aspect of a complete approach. Other aspects in order to investigate are for instance: user participation, the role of quality assurance and the selection of method fragments.

4. APPROACHES TO SYSTEMS DEVELOPMENT

During the field study at RCC the following basic approaches have been derived: phase-wise development, incremental development, stroke-wise development, evolutionary development and reverse development. A few supplementary approaches are recognized for the specific situation of RCC, namely: package development and package selection, the construction of components and development for re-use, experimental development and end-user computing. A supplementary approach cannot be used stand alone, but is always supplementary to a basic approach. An approach can be applied, if a number of pre-conditions, belonging to the approach, have been checked and satisfied. The sets of pre-conditions have been acquired by interviewing the project managers and experts of RCC. A specific questionnaire has been developed to support this research.

4.1 Phase-Wise Development

Figure 5. Phase-Wise Development

The main variant of phase-wise development is: strictly linear, phase after phase. This main variant is similar to the classical linear development model including some iteration, but the system does not return to the development cycle, except for maintenance. Such an approach does not allow intensive user participation and the application of formal planning and control techniques is usual. A disadvantage of this approach is the probably late detection of a possibly wrong route. Other variants are: subsystems tile-wise, subsystems in parallel, and the development of throw away systems. The variants "subsystems in parallel" and "subsystems tile-wise" are used to shorten the time needed for the development, depending on the availability of human resources. Pre-conditions for phase-wise development are:

- The specifications of the system are clear and stable. There is a clearly arranged project.

- How to realize the solution of the problem is clear and well-known. There is no uncertainty about the success of the project.
- Formal decision making is desired and necessary. The project must be controlled.
- It is a critical system with strategic importance for the customer organization.
- The system has a long lifetime. Low maintenance expenses are expected.
- The user and the developer have enough domain knowledge available.
- The developers have enough experience with appropriate methods, techniques and tools.
- It is not a small system.
- User acceptation of the system is not a problem.

4.2 Incremental Development

Figure 6. Incremental Development

If a part of the system has been developed, it enters the operations cycle. At the same time the development of another part of the system starts. Often, the nucleus of the system including one or more subsystems is developed firstly. After introducing that part of the system, the development of another part is started immediately. After realizing a part of the system, which must be integrated with other parts, it can be used by the user organization. The consequences for the user organization are less far-reaching and better manageable, because the changes are realized in small steps. A disadvantage may be that more time is needed for the development of the system in comparison with phase-wise development. However, increased quality, faster return-on-investment and the possibility to select the most urgent functionality may amply compensate this disadvantage. Pre-conditions for incremental development are:

- The timely delivery of the whole system is uncertain and the customer accepts a subset of the system on a certain date. After that date larger subsets of the system are delivered.
- Each time it requires a lot of work to introduce the realized part of the system. It must be necessary to have a working subset of the system very quickly.
- The specification of the first part of the system must be clear and stable.
- Through the realization of the first part of the system, one may expect to acquire more clarity about the other desired parts of the system concerning time for development, difficulties and required functionality. Usefulness and priority of the other parts of the system are discussed at the very beginning of the project.
- The quick delivery of a part of the system is required to get commitment and confidence of the users. Firstly, one must develop that part of the system, from which increased user participation can be expected.
- The system must be large enough to be divided into a number of large parts, that can be developed separately.
- If an incremental development strategy has been chosen for software package development, then the first version must offer enough functionality to get sufficient attention of the market.

4.3 Stroke-Wise Development

Figure 7. Stroke-Wise Development

One stroke of the development cycle corresponds to the development of a part of the system. The development of the whole system takes place through a number of subsequent development strokes (one time the development cycle) based on former development strokes. There exist two variants of stroke-wise development: aspect systems development and subsystems development. Aspect systems development is similar to throw away prototyping and subsystems development to keep-it prototyping. After every development stroke, it is checked whether the developed subsystem prototype satisfies the expectations and requirements of the users. The required changes are implemented during the next development stroke. The whole system is made ready for use during a final development stroke. Stroke-wise development is especially useful to reduce uncertainties, to evaluate intermediate products (subsystems) or to increase the involvement of users. Pre-conditions for stroke-wise development are:

- The specifications are unclear, because the users do not know exactly what they need.
- It is unclear how the problems should be solved.
- The business processes are not stable or not well-defined.
- Active user participation is necessary and the users must have enough time/capacity to evaluate the outcome of each development stroke.
- The users must be able to communicate about information needs and information models.
- The development organization must be able to demonstrate their expertise.
- Tools for a quick realization of prototypes must be available.
- Developers have little experience with the development of similar systems.
- The users are able to criticize the functioning of the prototype.

4.4 Evolutionary Development

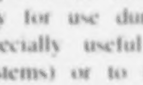


Figure 8. Evolutionary Development

During evolutionary development a complete system is developed, after which it is used. Based on experiences with the use of the system further development is undertaken, which means that the system evolves. Every version or release of the system is complete, which means that the user is provided with sufficient functionality for the time being. Only one delivery is planned for the current project, which differs from incremental development. A final number of releases is not determined beforehand. Each version of the system may be developed using a different development approach. One can use, for instance, for the first versions a prototyping approach to clarify the specifications, but later versions might be developed using a phase-wise approach, because the basic specifications of the system have already been clarified by using a prototyping approach for the first versions of the system. The pre-conditions for the evolutionary approach are:

- There must be enough clarity about specifications to develop the first version, or prototyping is used to clarify these.

- There must be a reason in order to believe that from the use of the system it will become necessary to realize important modifications in the next version. This may be the case if the system has an important impact on its environment, e.g. it changes the work procedures of the user, the interaction between systems development and organization development.
- Similar to incremental development, it must be possible to enforce the introduction of the system several times.
- Low maintenance and operations expenses must not be a requirement for the project.
- Through the use of the system new functionality must become emergent.
- This approach is also useful for package development. Assume that a package must be developed with complete functionality. Experience with the package in the market may lead to new functional specifications and critical remarks about the already realized functionality. Thereafter, the supplier of the package decides about the implementation of new or improved functionality in a new release.

4.5 Reverse Development

Figure 9. Reverse Development

RCC has not gained much experience with reverse development or reverse engineering. The expectation is that reverse development will become more important in the future. More often it is the case that a project does not start from scratch, but one starts from already existing systems.

The concept "reverse engineering" comes from hardware design, where the reconstruction of the design through analyzing the product is a common way of working. The same approach can be applied during information systems development by reconstructing higher level specifications from lower level specifications (binary code is the lowest level). This approach may simplify maintenance, the realization of modifications or replacement of components.

Re-engineering may be followed by a forward engineering step to change the functionality of the system, or not. Pre-conditions for reverse development are:
- Reverse engineering may be applied for well-functioning systems with a low technical quality, e.g. ill-structured systems.
- Reverse engineering is also a feasible approach, if we need documentation for maintenance purposes or for the construction of a new system and adequate documentation is not available.
- Computer Aided Reverse Engineering tools must be available.

So far, we described the basic approaches to systems development. The supplementary approaches can be described in the same way, but we wish to stop here, otherwise we get a too lengthy description. Of course, it is possible to construct hybrid forms of these approaches if the project situation makes it necessary. The idea is that each organization is able to build its own base of approaches using the models presented in this paper and learning from experience.

5. CONCLUSIONS AND FURTHER RESEARCH

We have exposed two ways for choosing a proper approach for information systems development. Firstly, based on a project characterization one can select or construct a proper approach. Secondly, starting from a set of well-known approaches one can check whether a certain approach matches the specific situation. The focus of this paper is on the second way of working. We have exposed a number of basic approaches, that appear to be useful in the case of RCC. Furthermore, a number of pre-conditions have been defined to check the situation for which the approach may be used. Of course, it is possible to define in practice variants of the defined basic approaches to achieve a better fit with the actual situation. The proposed meta process model is appropriate to derive from this model other situation-specific approaches. The process models and pre-conditions are developed by the organization dynamically, which means that within the organization arises an organizational memory, that can be consulted for similar situations. This process of consultation will be supported in the near future by computer aided method engineering tools (Harmsen et al. 1994).

Acknowledgements

We are very grateful to the staff of RCC. They spent a lot of time in facilitating the research and criticizing our reports. Also the critique of Dr. S. Brinkkemper was very helpful.

6. REFERENCES

Baskerville, R.J. Travis, D. Truex (1992), Systems Without Methods: The Impact of New Technologies on Information Development Projects, *IFIP 1992 Conference*, North-Holland.

Boehm, B.W. (1986), A Spiral Model of Software Development and enhancement, *ACM SIGSOFT software engineering notes*, vol. 11, no. 4.

Boehm, B.W. (1989), What We Really Need Is Process Model Generators, *Proceedings ACM*, p.397.

Burns, R.N. and A.R. Dennis (1985), Selecting The Appropriate Application, *Database*, Fall 1985.

Davis, G.B. (1982), Strategies for Information Requirements Determination, *IBM Systems Journal*, vol 21, no 1.

Harmsen, F., S. Brinkkemper, H. Oei (1994), Situational Method Engineering for Information System Project Approaches, To Appear in The Proceedings of IFIP WG 8.1, Maastricht.

Lyytinen, K. (1987), Different Perspectives on Information Systems: Problems and Solutions, *ACM Computing Surveys*, 19.

Naumann, J.D. et al. (1980), Determining Information Requirements: A Contingency Method for Selection of a Requirements Assurance Strategy, *The Journal for Systems and Software*, 1.

Ould, M.A. (1990), Strategies for Software Engineering: The Management of Risk and Quality, *Wiley Series in Software Engineering Practice*.

Schoonhoven, B. (1993), Internal Technical Report on Situational Approaches (Dutch), University of Twente.

Slooten, C. van, S. Brinkkemper (1993), A Method Engineering Approach to Information Systems Development, *Information Systems Development Process*, IFIP WG 8.1, North-Holland.

Slooten, C. van, S. Brinkkemper, P. Hoving (1994), Contingency Based Situational Systems Development in Large Organizations, *Managing Social and Economic Change with Information Technology*, Idea Group Publishing, Harrisburg.

BOOM - A First Step to an Object-Oriented Fourth Generation System

Hubert Rumerstorfer, Josef Altmann

Management Information Systems, Software Engineering Department
Johannes Kepler University, A-4040 Linz, Austria
Tel.:++43 (70) 2468-9439, Fax: ++43 (70) 2468-9430
Email: {rumerstorfer, altmann}@swe.uni-linz.ac.at

Abstract

This paper describes BOOM, an object-oriented 4th-generation system. BOOM supports the powerful and elegant development of information systems in an object-oriented manner. By using object-oriented techniques the complexities in building information systems can be handled more easily and the development process in general can be improved and revived. BOOM decouples development from the physical environment. BOOM developers build a logical object-oriented data model that, after development, is automatically transferred into a relational database system. The toolset supplied for the construction of information systems includes components for the definition of structural object-oriented data models, for the generation of user interfaces, and the arrangement of application logic. Generic operations for manipulating user-defined objects are provided for every BOOM application. BOOM objects are stored in a relational database and are dynamically loaded on demand. Further, how the object model is transformed to relational tables is illustrated.

Introduction

Fourth generation systems have proven to be powerful tools for the development of information systems. Many different names have been suggested for this same sort of development systems: Application generator, database management system and database-oriented development tool often serve as synonyms for 4th-generation systems. The common goal to all these systems is to provide the programmer with tools and functionality for every phase of database application development, making programming tasks easier, quicker, and safer by raising the abstraction level.

Most of the 4th-generation systems are designed for a particular database system, especially relational databases, and can therefore take maximum advantage of all the system's features. These systems differ widely in their universality and applicability: Some of them can only be handled by professional developers, whereas others are suitable for "programming" by end users.

Terminology

We consider 4th-generation systems as a collection of common and basic features which are typical for such development environments (see also Martin, 1985). The most important and prevailing components provided by these systems are:

- a graphical editor for the specification of the data schema based on some form of a relational model
- a layout/forms editor for the design of user interfaces, screen, and print reports
- a menu editor for the interactive definition of menus and their commands
- a procedure editor to write code with an embedded programming language; little chunks of code can be attached directly to user interface elements, a process called scripting (Goodmann, 1987)
- a tool to supply the application with context-based help
- a query component for retrieving and sorting data according to desired criteria

Some 4th-generation systems additionally supply a comfortable graphical tool for specifying database tables, columns or other criteria that describe the data to be searched for. This tool can then automatically generate the required statement to get the data from the back-end database to the application.

As a further requirement, a 4th-generation system should support the linking to standard applications (word processors, spreadsheets, etc.) or custom code implemented with languages other then the database application. The following Figure 1 shows typical activities during the information systems development with a 4th-generation system and the respective tools.

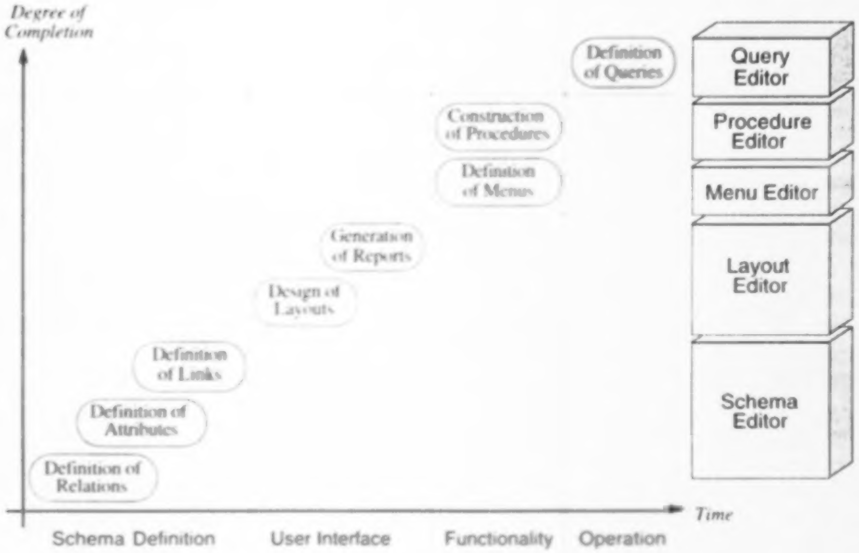

Degree of Completion
Definition of Queries
Query Editor
Construction of Procedures
Procedure Editor
Definition of Menus
Menu Editor
Generation of Reports
Layout Editor
Design of Layouts
Definition of Links
Schema Editor
Definition of Attributes
Definition of Relations
Time
Schema Definition
User Interface
Functionality
Operation

Fig. 1: Typical development cycle with a 4th-generation system

To justify the word *object-oriented* in conjunction with 4th-generation, the system has to provide (besides the typical features like data definition editors, layouting tools and generators) richer possibilities for the modeling of entities. The object-oriented implementation alone does not suffice to call a 4th-generation system object-oriented.

Current 4th-generation systems on the market are *relational* in the sense that they only support the modeling of real-world entities into relational tables. An object-oriented 4th-generation system has to provide an object model, i.e. the posited characteristics such as encapsulation, inheritance, and polymorphism have to be supported.

Recent 4th-generation systems that include object-oriented features support aggregation and delegation of functionality for user interface objects. In some systems these mechanisms are called component inheritance and polymorphism (e.g., a label in a button in a window can inherit from the window). This view of inheritance may be simpler, more intuitive, and more useful for the user interface construction. But it should not be confused with a type (is-a) inheritance hierarchy because it lacks the semantic power of an usual class (is-a) inheritance hierarchy and it does not touch the data model of the system.

Our goal in developing the BOOM system therefore has been to provide the user with simple but yet powerful tools to create information systems. The advantages of object-oriented design are used for modeling the real-world entities in this area. Flexibility is achieved due to the use of object-oriented programming, which also makes it possible to extend the development system itself. Nevertheless, we do not disregard the benefits of current relational database technology and incorporate it into our tool.

The construction process of information systems

The traditional way to develop database applications is first to design the data model (the logical and physical data structures) and then to define screen layouts, menus and the application logic. For a long time this was seen as the engineering procedure in information systems development.

With the introduction of workstations with high-resolution graphics and new programming environments, a new approach for the development of highly interactive software arose. A new method, *prototyping*, was launched to give a scientific background to this new approach model. New easy-to-use and powerful tools allowed us to paint and experiment with screen layouts before we actually defined a data model. VisualWorks™ (ParcPlace, 1992a), for example, uncompromisingly follows this new pure prototyping approach: first the interface (with all necessary fields, list, buttons, menus, etc.) is designed and afterwards the class definition and stub methods for actions are generated by the system.

Our system avails the user both possibilities. With the traditional approach the object model has to be designed first (necessary class definitions are generated). A default user interface, which can be adjusted with some options in the layouting, is generated and a generic framework for testing/running the application is provided. The prototyping approach uses the VisualWorks™ user interface painter in the first place and lets VisualWorks™ generate the necessary classes and accessing methods.

The Object Model of BOOM

An important goal in the development of our system has been to provide the designer of an information system with modeling concepts that are as simple and as easy to learn as possible. On the other hand, the object model has to be rich enough to fulfill the requirements for the modeling of typical applications in the information systems field (Won, 1990).

In the words of Adele Goldberg (1984): "The problem of creating a friendly programming environment centers on the kind of help the system provides, and the ease with which we can cause the effect we wish to cause." The point is that the friendliness is quantified by the distance between what the user has in mind and how he can do it with the system. The object-oriented approach decreases this distance by the everything-is-an-object philosophy and so provides a more natural view of software components.

Our approach aimed to define an appropriate data model which is keyed to the development of information systems and which supports the modeling of user-defined objects and relationships between them. In various investigations on the implementation of information systems, we have realized that above all a structural object model is necessary. The behavioral aspects in information systems are very often of a generic nature (which is common to all object types) and can be provided by the system in advance.

The first step in the construction of an application is to define object types which share the same structure. An object type is described by the definition of attributes, which are either atomic (character string, integer, float, date, etc.) or complex (a previously defined object type). Atomic attributes are directly mapped to database fields. Besides, an attribute may be multi-valued, which means that it holds a collection of values of a defined type. Because it is not always useful and desirable to have a complex attribute fetched automatically from the database when the object is loaded, there is the possibility to turn off the automatic linking to these complex components. They can be loaded on demand.

To guarantee data consistency and ease the data entry plausibility checks, constraints, rules or a choicelist of possible values may be defined for certain attributes. When defining a new object type there is the possibility to start from scratch or to inherit the structure from a previously defined object type (e.g., Employee inherits attributes from Person).

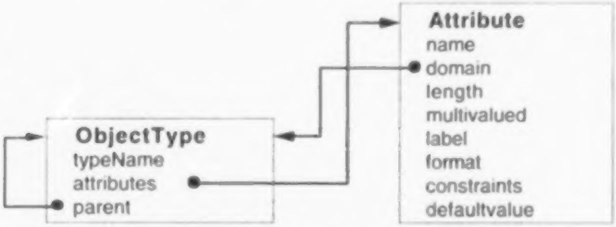


Fig. 2: Object Model of BOOM

In Figure 2 the core structure of object types is depicted in a class diagram. The parent reference indicates that an object type can be derived from another one. Each object type is characterized by a set of attributes. The domain of an attribute can be of a scalar data type (character string, integer, etc.) or a reference to a previously defined object type.

For the user of BOOM there is no need to define an identifying key. An object identifier is automatically attached to each object, and created if necessary by the system. In a running system it is often necessary to identify an object by one attribute (e.g., selecting persons from a list with first and last names). *Therefore each object type has a certain attribute* marked as representative for the object, but it need not be unique.

When defining a complex attribute, we have to distinguish between a component attribute (a real part, which belongs exclusively to the object) and a reference attribute (a description, date or resource which is shared by other objects). References are also called usage relationships. The definition of reverse relationships (e.g., an employee holds a reference to a department and a department has references to employees) is possible and detected by the system when the application is set up and the table definitions are generated.

The Toolset of BOOM

The selection of tools provided by BOOM can be compared to traditional 4th-generation systems. The real difference lies in its potential and possibilities, especially the modeling component. Due to the object-oriented implementation, it is easily possible to extend the development system itself or to exchange certain tools. Figure 3 shows the available set of tools in the BOOM Launcher. The BOOM system has been developed using the class library of Smalltalk-80 VisualWorks™ (ParcPlace, 1992a). The user interface framework of VisualWorks™ is very powerful and has been integrated for the user interface component of BOOM. Because the layouts of VisualWorks™ itself have been developed using the VisualWorks™ user interface builder it is easily possible to adapt the screen painting possibilities for the BOOM environment.

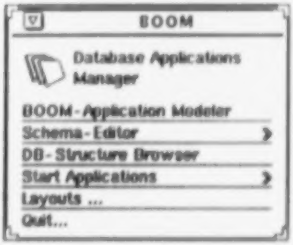

BOOM
Database Applications Manager
BOOM-Application Modeler
Schema-Editor
DB-Structure Browser
Start Applications
Layouts ...
Quit...

Fig. 3: The BOOM-Tools Launcher

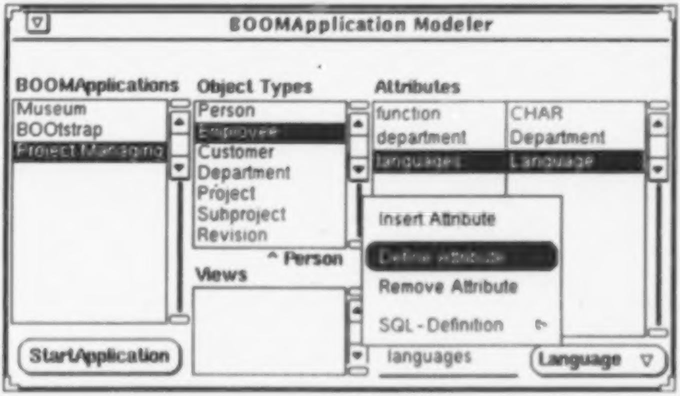
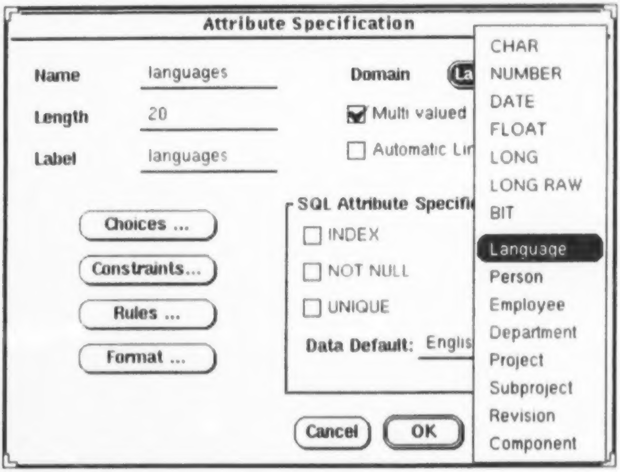

BOOMApplication Modeler
BOOMApplications
Museum
BOOtstrap
Project Managing
Object Types
Person
Employee
Customer
Department
Project
Subproject
Revision
^ Person
Views
Attributes
function
CHAR
department
Department
languages
Language
Insert Attribute
Define Attribute
Remove Attribute
SQL-Definition
StartApplication
languages
Language

Fig. 4: The BOOM Application Modeler

The main component is the BOOM Application Modeler (Figure 4). This tool supplies a browser-oriented interface for the definition of applications and their object types.

With the definition of the object types and their relationships essential for a running application, the main work is done. A sample dialog for the definition of an attribute of a certain object type is shown in Figure 5.



Attribute Specification

Name: languages
Length: 20
Label: languages
Domain: La
☑ Multi valued
☐ Automatic Lin
SQL Attribute Specifi
☐ INDEX
☐ NOT NULL
☐ UNIQUE
Data Default: Englis

Fig. 5: Attribute specification

The definitions of corresponding classes and database tables are automatically generated. For the generation of a default entry layout several patterns are available. The adapted user interface painter can be used to enhance the layouts or adjust them to given requirements. Additional tools like menu or mask editors of VisualWorks™ can be used if necessary.

A second, more comfortable way to construct the schema for a certain application is to use the schema editor, which has been developed using the framework vis-A-vis for graphical design tools (Lichter and Schneider, 1993).

After the selection of an application, a window with the graphical model of the application appears. The graphical notation bases upon the structural class diagrams suggested by Wilson (1990).

For tuning the relational database, we have implemented a tool called DB-Structure Browser, which enables the user to see the actual relational structure and to optimize the database operations by defining additional indices and clusters on columns.

Connection to a relational database

To make objects persistent or to store data from an object-oriented framework like Smalltalk-80 has always been a challenge for object-oriented application developers.

As a matter of fact, a lot of object-oriented application designers face the issue of integrating objects with relational technology. Due to the widespread use of relational databases, the relational and the object technologies will need to coexist for a long time into the future (Loomis, 1991).

Relational applications and database management systems will persist alongside applications implemented with object-oriented programming languages (like C++, Smalltalk, etc.) and accessing object-oriented databases. The demand is to be able to build object-oriented applications that access existing relational databases.

Besides these pragmatical reasons, the support provided in existing relational databases (e.g., concurrency, security, portability) encouraged our decision to combine relational and object technologies. Our approach tries to build new applications using object-oriented programming and to access relational databases using the BOOM interface (Figure 6).

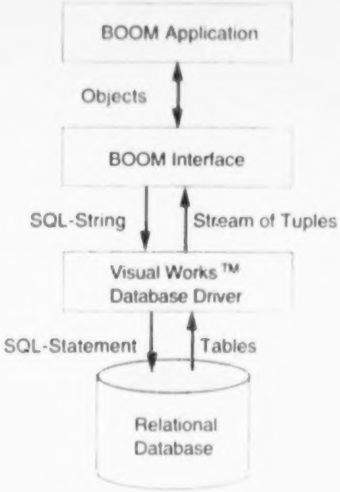


Fig. 6: Database-Gateway

The BOOM system implements a Database Interface Generator, which allows a program constructed with BOOM to access the relational database (e.g., Oracle®, Sybase®) as if it contained object instead of tables. The BOOM programmer does not need to know how objects are stored as table rows. In other words, it is not necessary to use SQL when dealing with the relational database. Furthermore, the BOOM interface translates the object operations into SQL strings which are built up dynamically, and transposes the resultant streams of tuples into objects based on the knowledge of the mapping between BOOM's object model and the relational database schema. The BOOM interface provides generic functions for loading, manipulating, and querying objects whose structure has been defined with the BOOM Application Modeler.

Mapping BOOM Object Classes to Tables

Relational database systems provide the user with a view of data as a collection of tables. Such a table represents an entity type and the columns define the attributes. The individual rows (tuples) of a table represent instances of an entity type.

In this section, we consider some mapping rules for classes onto tables as described by Jacobson (1992) and Rambaugh et al. (1991). We mainly followed these in the implementation of BOOM.

First of all it is necessary to decide which classes and which variables of a class have to be stored in the database. Each object class could be represented by a table in the database. By using a relational database, we have to take care of transforming the class definitions into a data model consisting only of flat relations. In the simplest approach, each table definition corresponds to a class, each column corresponds to an attribute, and each table row corresponds to an instance of a class. Simple classes containing attributes with standard basic data types can simply be transformed into flat relations.

One of the limitations of relational systems is the restricted number of data types, such as integer, floating-point number, string, date and time. In this case, relational databases are not designed to manage arbitrary user-defined data types. If there are complex user-defined data types which are composed of basic, the definitions of such variables have to be analyzed and decomposed to atomic attributes of the relation. The following mapping rules describe the transformation process in the BOOM interface:

- Each class declaration maps to one or more tables. In the case of a complex object structure which holds references to other object types, a class corresponds to more than one table.
- Each (primitive) instance variable defined in a class becomes one column in the table. In the case of a complex instance variable, an additional table is attached.
- Each table derived from a class has an identifier for the primary key; one or more object identifiers create the primary key for association-derived tables.
- A row in an class-derived table represents an instance of the class.
- Mapping associations with a cardinality greater than one leads to a new table. Therefore foreign keys are used as surrogates to represent a relationship between two tables.

The notions of inheritance and generalization are not supported in relational database systems. For mapping inheritance to a relational database, Jacobson (1993) suggests two principally different approaches:

- There is one table for each class which contains all inherited attributes. Therefore inherited attributes from the superclass are copied to all the tables that represent a subclass. Consequently, no table represents the abstract class, and at the same time the hierarchical inheritance is gone.
- The superclass represents one table, to which the tables of the descendant classes refer. This means that if the class is an inherited class, the matching table contains only the instance variables which are newly introduced in the definition of this class. For that reason it is necessary to establish a reference to the table of the superclass.

The BOOM interface implements the second approach, where the superclass and each subclass map to a table and the attributes of the superclass table are not replicated for each subclass table.

There are many other possible mapping strategies between tables and class structures (Daniels and Cook, 1993). Some of these mappings could be done automatically; others require detailed application knowledge. However, the problem of low performance often occurs as joining and searching in several tables is necessary to get information about one object. The DB-Structure Browser of BOOM gives you the possibility to optimize such relational operations.

Architectural Overview

The overall design of the BOOM framework is depicted in Figure 7. It shows the most important classes together with their relationships. We concentrate on clarifying the architectural parts that are directly involved in the execution phase of a BOOM application.

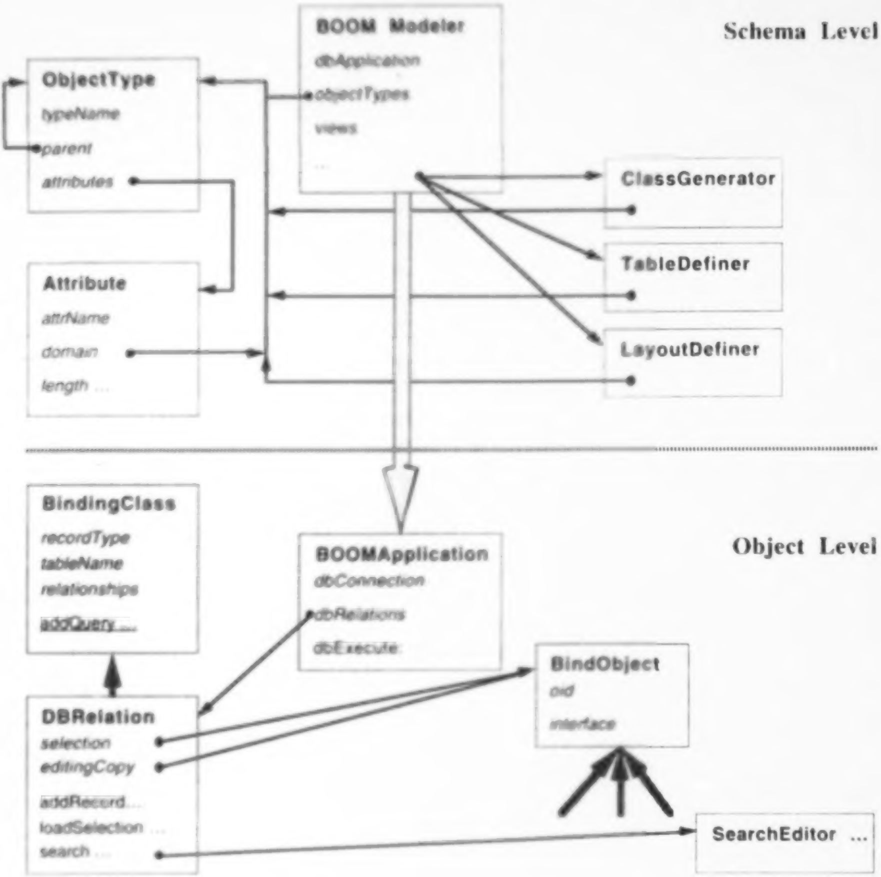

BOOM Modeler
dbApplication
objectTypes
views
...
Schema Level
ObjectType
typeName
parent
attributes
ClassGenerator
TableDefiner
Attribute
attrName
domain
length ...
LayoutDefiner
BindingClass
recordType
tableName
relationships
addQuery ...
BOOMApplication
dbConnection
dbRelations
dbExecute
Object Level
BindObject
oid
interface
DBRelation
selection
editingCopy
addRecord...
loadSelection ...
search ...
SearchEditor ...

Fig. 7: Principal architecture of BOOM

When working with BOOM we can distinguish between two levels. In the schema level we construct an application by defining object types.

When starting an application, we change to the object level. For every object type, a subclass of BindObject is generated. BindObject implements the abstract protocol for objects which are manipulated by generic operations, like adding, deleting, updating, sorting, and searching, of the class DBRelation. BindingClass holds the intentional description of these objects; DBRelation keeps the extension of these object types.

Finally, we note that the construction of a new BOOM application consists mainly in building subclasses of BindObjects.

Conclusion and Perspectives

There are a lot of 4th-generation systems on the market, but only a few of them take advantage of object-oriented programming. BOOM integrates a toolset to a universal application framework. With object-oriented modeling techniques the process of building information systems can be dramatically improved because real world objects are directly represented in the program. Because no considerations about the transformations into relational tables are necessary, the development process is accelerated. Besides, it is widely accepted that object-oriented programming leads to more elegant program structures. Due to inheritance, generic operations which are very typical in the information systems domain can be extracted to a superclass. New functions which are not directly supported (as generic operations) can easily be added by subclassing.

The BOOM system is still evolving, and new concepts and features are added regularly. In a bootstrapping approach we constructed a BOOM application which defines the object model of BOOM and translates it into tables of the relational database. In this way the schemas of the BOOM applications are made persistent in the same way as the application data itself. Our next step in this direction is to define a meta-database which describes in a universal way the conversion process from the object model to relational tables.

The object model itself will also be enriched by providing view definitions. These views show only some aspect of an object type or a collection of several types and can be seen as virtual object types. Another promising concept for construction of information systems is the role model in conjunction with objects (Gottlob et al. 1994).

Literature

Daniels, J., Cook, S., (1993), "Strategies for sharing objects in distributed systems", Journal of Object-Oriented Programming, Vol. 6, Nr. 1, pp. 27-36.

Gamma, E., (1992), "Objektorientierte Software-Entwicklung am Beispiel von ET++", (in German), Springer, Berlin.

Goldberg, A., (1984), "The influence of an object-oriented language on the programming environment", In interactive programming environment, McGraw-Hill.

Goodman, D., (1987), "The Complete HyperCard Handbook", Bantam Books, New York.

Gottlob, G., Schrefl, M., Roeck, B., (1994), "Extending Object-Oriented Systems with Roles", ACM Transactions on Information Systems, submitted for publication.

Jacobson, I., (1993), "Object-Oriented Software Engineering", Addison-Wesley, Reading.

Lichter, H., Schneider, K., (1993), "vis-A-vis: An Object-Oriented Application Framework for Graphical Design Tools", Proc. of IFIP Workshop on Industrial Systems, Darmstadt, Elesevier.

Loomis, M.E.S., (1991), "Object and SQL: Accessing relational databases", Object Magazine, Vol.1, Nr. 3, pp. 68-78.

Martin, J., (1985), "Fourth Generation Languages", Prentice Hall, Englewood Cliffs.

Oracle Cooperation, (1990), "Oracle® Database Administrator's Guide", Redwood Shores.

ParcPlace Systems Inc., (1992a), "ObjectWorks®\Smalltalk™ Release 4.1, Users Guide", Sunnyvale.

ParcPlace Systems Inc., (1992b), "VisualWorks™ Release 1.0, Users Guide", Sunnyvale.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W. (1991), "Object-oriented modeling and design", Prentice-Hall, Englewood Cliffs.

Wilson, D.A., (1990), "Class Diagrams: A Tool for Design, Documentation and Testing", Journal of Object-Oriented Programming, Vol. 3, Nr. 1, pp. 38-44.

Won, K., (1990), "Introduction to object-oriented databases", MIT-Press, Cambridge.

Won, K., (1991), "Object-oriented systems: strengths and weaknesses", JOOP, Vol. 4, Nr. 4, pp. 21-29.

INTEGRATION OF STRUCTURED METHODS AND FORMAL NOTATIONS

Pat M. Allen[1] and Lesley T. Semmens[2]

[1] University of Huddersfield, School of Computing and Mathematics, Queensgate, Huddersfield, HD1 3DH, England

[2] Leeds Metropolitan University, Faculty of Information and Engineering Systems, The Grange, Beckett Park, Leeds, LS6 3QS, England

Abstract

Structured software development methods provide means of managing the complexity of large systems and have gained wide acceptance during the last decade. In parallel with this formal languages and methods have been developed. This paper reports on and describes recent work which demonstrates the benefits of integrating the two. Two general approaches are outlined: one pragmatic and the other fully integrated. The integration of Yourdon and Z is presented as a case study. A to[illegible] which has been developed to support this integration is described.

1 Introduction

The 1980s saw a large increase in the use of "structured" methods such as Yourdon Structured Analysis (Yourdon, 1989), SSADM (SSADM, 1990), Merise (Rochfeld et al., 1983) and Object Oriented Analysis (Shlaer and Mellor, 1988). Such methods have achieved widespread use because they offer a number of advantages to the developer of large information systems. They offer a structured approach, providing a means to structure and manage the complexity of large systems. They provide the analyst/designer with a variety of modelling techniques, such as entity-relationship modelling, data flow modelling, entity life histories and so on, generally expressed in now-familiar graphical notations. They are widely used and well supported with CASE tools, training etc. They provide an essential framework for project management, dividing the development task into stages, with well-defined deliverables at each stage.

However there are disadvantages. The notations used lack formal semantics, so it is not possible to give a definitive interpretation of a specification, or ensure that it means the same thing to the author and the reader. There are no reasoning mechanisms, so there can be no formal procedure to ensure that the specification is free from inconsistencies, or that it meets particular meta-requirements, pertaining perhaps to security or data integrity.

Formal methods such as Z (Spivey, 1992), VDM (Jones, 1986) and RAISE (Neilsen et al., 1987) use mathematical languages which are unambiguous, and allow the requirements for systems to be specified with great precision. This precision can be achieved while still maintaining a high level of abstraction — a significant advantage. There is a growing body of evidence that the main benefit of formal methods lies in the acheivement of a precise, unambiguous specification of requirements, and in the rigour with *which the analysts must approach their task in order to produce one, and* this certainly need not only apply to safety-critical or highly secure systems.

Of course, formal methods too have their disadvantages. For a start, most are not really methods, in that they do not have procedures and stages for carrying out software development tasks. Even those formal methods such as VDM which can justifiably lay claim to the title really only address relatively small scale design problems. They are hard to "scale up" for real-life projects. The mathematical notations themselves are unfamiliar to many analysts and software engineers and there is a lack of experienced personnel. Although some tools have been developed there is still not a large enough market to justify large scale investment by major vendors.

We consider structured and formal methods to be complementary approaches, and see no fundamental reason why the advantages of both should not be combined in order to improve the quality of our information systems specifications.

Over the last few years there has been something of an explosion of research interest in the integration of structured methods and formal notations, and in the next section we give a brief survey of some of the most important work.

2 Approaches to methods integration - the state of the art

In a survey of current research and practice it is possible to identify two general approaches. The first is essentially pragmatic, using a structured method to define modularity, and then a mathematical notation to formally specify the modules. This is type of approach has been used, for example, by Rolls Royce and Associates, who used Yourdon together with VDM (Hamilton, 1991), and Praxis, who used Yourdon and Z (Hall, 1992). Both of these users report considerable success with this type of approach, and it probably characterises most current industrial experience using structured and formal methods on the same project.

The second general approach seeks a closer integration between the stuctured method and the formal notation, and this is achieved by formalising the notations of the structured method. The products of the structured method, entity-relationship diagrams, data flow diagrams etc. then become formal objects with a well defined semantics. Translation (possibly automatic) can be performed between the structured notations and the mathematical language, and formal checking for consistency between the formal and structured specifications also becomes a possiblity.

Examples of the "integrated" type of approach include our own work on the integration of Yourdon with Z (Semmens and Allen, 1991a) (described in the next section), and the work of Larsen et al. at the University of Delft, who have described a method which integrates SA/SD with VDM (Plat, 1993). (Aujla et al., 1993) reports a rigorous review technique for SSADM specifications, using Z.

This survey is far from exhaustive; for a more complete review see (Semmens et al., 1992).

3 Yourdon/Z: a case study in method integration

3.1 A scheme for method integration

In order to test the viability of an integrated approach we chose to look at Yourdon's Modern Structured Analysis method (Yourdon, 1989), and the Z notation. Yourdon was chosen because it is widely used, because it is well provided with cross-checking mechanisms between the documents produced using its various techniques, and because it uses event-response partitioning which we believed made it particularly well suited for use with a model-based formal method.

A Z specification normally consists of a mathematical model of the system state, constructed using mathematical structures such as sets, relations and functions, and a number of abstract operations, defined by specifying their effects on the abstract state. The specification of the system state can express all of the information contained in a Yourdon ERD (and additional requirements about the state data — Z is a more expressive notation than ER diagrams). The abstract operations are exactly the system functions, which in turn are the responses to the external and temporal events to which the system must respond. These correspond exactly to the processes on the Yourdon event-level data flow diagram.

Three stages were necessary in the development of an integrated method. Firstly a method framework was specified, showing the stages in the process with clearly defined deliverables at each stage. The framework retains much of the orginal Yourdon method, but includes some modified or additional stages to complete the formal specification. Next a formal semantics for the Yourdon notations was defined, in order to formalise the products of the structured methods (diagrams, structured text). Finally, because we believe that the integrated method could not realistically be used in practice without CASE tool support a specification of the required software tools was drawn up. In fact the formal semantics defined for the Yourdon notations became the specification for an important component of an integrated tool set.

3.2 The Yourdon/Z method

For the initial analysis of requirements the method follows the stages prescribed by Yourdon, as far as the production of an event-level data flow diagram specifying responses to the external and temporal events. These stages will produce the following deliverables: Context Diagram, Event List, Entity Relationship Diagram (ERD), Event Level Data Flow Diagram (DFD), Data Dictionary (DD).

Next the structured specification is translated into Z. Z specifications are structured using schemas, which provide modularity. Typically a schema, or possibly a set of schemas, is used to specify the system state, and a schema is used to specify each abstract operation. In Yourdon/Z, the ERD and DD are transformed to Z state schemas, and partial Z operation schemas are derived from the DFD.

Finally the formal specification is completed. Any additional constraints on the state are added, and operation schemas are completed by formally specifying what are effectively their pre- and post-conditions.

The main stages of the method are summarised in Figure 1.

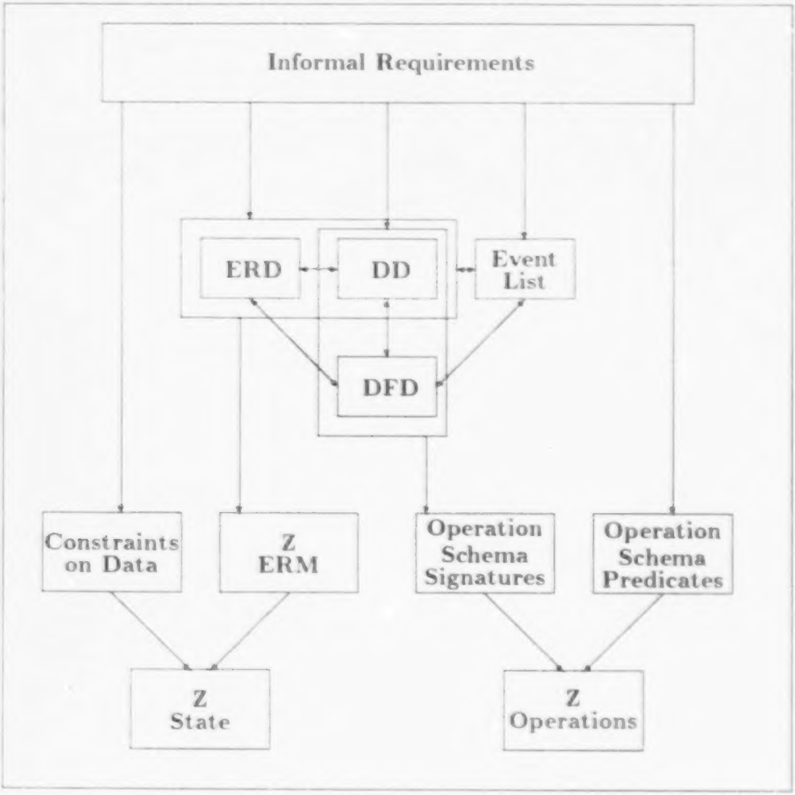

Informal Requirements
ERD
DD
Event List
DFD
Constraints on Data
Z ERM
Operation Schema Signatures
Operation Schema Predicates
Z State
Z Operations

Figure 1 : The Yourdon/Z Method

The result is a structured and formal specification of the system comprising: Context Diagram, ERD, DFD, DD, Z state and operation schemas.

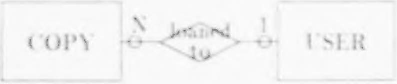
3.3 A semantics for the Yourdon notations

3.3.1 Formalising the structured notations

In our study, each of the Yourdon notations mentioned above was formalised as follows. First an abstract syntax was defined (in Z) for the structured notation. Next an abstract syntax was defined for that subset of the Z language needed to express the meaning of the structured specification. Finally a set of translation functions were defined (also in Z) mapping the syntax of the notation to that of the Z subset. Since Z already has a denotational semantics (Spivey, 1988), this was equivalent to defining a formal semantics for each of the Yourdon notations. The formal semantics can be found in a technical report (Semmens and Allen, 1991b). In this section we will just give a few simple examples of structured notation with their Z equivalents. A realistic example of the use of Yourdon/Z (to specify the data security subsystem of a large information system) can be found in (Semmens et al., 1990).

3.3.2 The data model

Let us first consider an extract from the data model for a library system, showing two entities and a relationship between them:



There will be associated data dictionary entries:

COPY ::= @COPYNO + TITLE + SUBJECT + LOCATION
USER ::= @USERID + NAME

In Z we would first define *basic types* (given sets) for the types of the entity attributes:

[*COPYNO*, *TITLE*, *SUBJECT*, *LOCATION*]
[*USERID*, *NAME*]

and the entity types can then be represented by schemas which just contain variable declarations:

Copy
copyno : *COPYNO*
title : *TITLE*
subject : *SUBJECT*
location : *LOCATION*

User
userid : *USERID*
name : *NAME*

We can define schemas to represent the sets of instances of the entities in the system. These correspond to the data stores on the event level DFD. As well as declarations (the *signature* of the schema), a Z schema can include a predicate which constrains the possible values of the variables. In *Copy_ds* and *User_ds* an injective function from the entity type to the type of its key attribute, with suitable constraints on its value, ensures the uniqueness of the keys.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Copy\_ds}} \\ \hline
copies : \mathbb{P}\ Copy \\
copy\_id : Copy \rightarrowtail\!\!\!\!\!\!\!+\;\; COPYNO \\ \hline
\mathrm{dom}\ copy\_id = copies \\
\forall c : copies \bullet copy\_id(c) = c.copyno \\ \hline
\end{array}$$

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{User\_ds}} \\ \hline
users : \mathbb{P}\ User \\
user\_id : User \rightarrowtail\!\!\!\!\!\!\!+\;\; USERID \\ \hline
\mathrm{dom}\ user\_id = users \\
\forall u : users \bullet user\_id(u) = u.userid \\ \hline
\end{array}$$

A relationships between entities can be expressed mathematically as a binary relation. Since the relationship between copies and users is many-to-one, we can use the Z notation for a partial function.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Loaned\_to\_ds}} \\ \hline
loaned\_to : Copy \rightarrow\!\!\!\!\!\!\!+\;\; User \\ \hline
\end{array}$$

Suitable translations have also been defined for entities with subtypes and for associative entities.

When all of the entities, relationships and their instance sets have been defined a schema can be defined which represents the entire system state. We can use *schema inclusion* to obtain all the declarations and invariants already specified in *Copy_ds*, *User_ds* etc. One of the advantages of Z is its modularity, and in fact it provides a powerful calculus for operating on and combining schemas. We have used an ERD notation which specifies whether participation by the entities is optional or compulsory. Mathematically the same information can be given by placing suitable constraints on the domains and ranges of the relations and the entity instance sets.

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{Library}} \\ \hline
Copy\_ds \\
User\_ds \\
Loaned\_to\_ds \\ \hline
\mathrm{dom}\ loaned\_to \subseteq copies \\
\mathrm{ran}\ loaned\_to \subseteq users \\ \hline
\end{array}$$

3.3.3 The process model

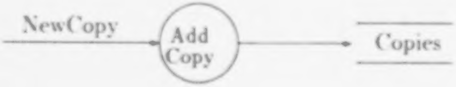
Having performed the translation and obtained this state schema, we could add additional information from the requirements, which cannot be expressed in the entity-relationship notation. For example, suppose that there was an upper limit on the number of books each user could borrow. This could be expressed in Z as follows:

$MaxCopies : \mathbb{N}$

Library
Copy_ds
User_ds
Loaned_to_ds

$\mathrm{dom}\ loaned\_to \subseteq copies$
$\mathrm{ran}\ loaned\_to \subseteq users$
$\forall u : \mathrm{ran}\ loaned\_to \bullet \#(loaned\_to \rhd \{u\}) \leq MaxCopies$

Having specified the system state, some information about the operations on that state can be obtained from the event level DFD. Let us consider an extract from that DFD which shows a process which updates the relevant data store when a new copy is added to the library:



The contents of the data flow *NewCopy* can be defined by a schema:

NewCopy
$title : TITLE$
$subject : SUBJECT$
$location : LOCATION$

We can then write the signature of a Z operation schema:

AddCopy
$\Delta Library$
$\Xi User\_ds$
$\Xi Loaned\_to\_ds$

$NewCopy?$

Conventionally, input variables are "decorated" with a question mark. The Δ notation indicates that the operation causes a state change. The Ξ notation is used to show that the *User_ds* and *Loaned_to_ds* variables specifically are not changed by adding a copy.

Of course as it stands the schema does not specify what the operation actually does. That will be done by adding a predicate to the schema specifying the necessary constraints on the before and after state variables, inputs and outputs, if any. These complete operation schemas replace the Yourdon process specifications, which typically are expressed using Structured English. Z operation schemas provide much more abstract and declarative specifications of the requirements for the system functions. This is a good example of how formal methods can provide precision while maintaining a high level of abstraction.

3.4 Tool support

3.4.1 Generic tool types

The tools which are currently available to support Yourdon/Z fall into three categories. There are structured method support tools, formal methods support tools and a new class of tool which we have called a "specification assistant". Our current support environment includes some tools developed in-house, and some from outside vendors.

3.4.2 Structured method support tools

These are CASE tools of the type used to support structured methods such as Yourdon or SSADM. They include graphical and text editors to enable users to produce the documents and diagrams required by the method, and checking facilities to ensure that the rules of the method have been followed and that the specifications produced are consistent.

3.4.3 Formal method support tools

We currently use J.M. Spivey's *f*UZZ tool, which provides syntax and type checkers for Z specifications.

3.4.4 Specification assistant

This is a completely new type of tool, which we believe is needed to support the use of integrated formal and structured methods. The specification assistant is an integrated toolset providing three main functions. First of all it translates the diagrams and structured text documents into a formal notation (or, in the case of our tool, allows a formal representation of the stored data about the system requirements). It also provides an editor to allow the formal specification to be extended or altered. Finally, it provides consistency checking to determine whether a formal specification is consistent with the results of the analysis produced using the structured method.

A working prototype of a specification assistant tool has been developed for Yourdon/Z, which is fully integrated with the structured method support tools.

4 Conclusions

Our survey of recent work shows that the integration of formal and structured methods is gaining widespread interest, both in research and in industrial practice.

We have identified two different approaches:

Use structured method to define modularity, then specify modules formally.

Formalise the notation of the structured method.

As a case study in the application of the second of these approaches, we have defined a method which integrates Yourdon Modern Structured Analysis with the Z notation.

This case study has shown that it is possible:

1. To formalise completely the notations of a structured method suitable for the systematic development of information systems
2. To develop a requirements analysis method with well-defined stages and deliverables, which uses structured techniques and formal notations in an integrated way, and which produces a specification that is both structured and formal
3. To provide tool support for such an integrated method.

Although Yourdon/Z was developed as a demonstration of a particular approach to formalisation, rather than a commercial method, it has been applied to a number of non-trivial systems with encouraging results.

Current and future work includes the development of a specification method based on integrated formal and object-oriented methods, and a software engineering environment to support the use of integrated methods.

References

Aujla,S., Bryant,T. & Semmens,L., (1993), A rigorous review technique: using formal notations within conventional development methods, Software Engineering Standards Symposium, Brighton, UK, August/September 1993, IEEE Computer Society Press.

Hall,A., (1992), Development of an Air Traffic Control System, at BCS/FACS Workshop, Imperial College, London, December 1992.

Hamilton,V., (1991), Experiences of combining Yourdon and VDM, at Methods Integration Workshop, Leeds, September 1991.

Jones,C.B., (1986), Systematic Software Development Using VDM, Prentice-Hall.

Neilsen,M., Havelund,K., Wagner,K.R. & George,C., (1987), The RAISE language, methods and tools, in "VDM'88 Symposium", Lecture Notes in Computer Science 328, Springer-Verlag.

Plat,N., (1993), "Some Experiments with Formal Methods", PhD Thesis, Technical University of Delft, The Netherlands.

Rochfeld,A. & Tardieu,H., (1983), Merise: An information system design and development methodology, *Information and Management*, Vol 6, pp 143-159.

Semmens,L.T., Allen,P.M. & Evans,A.E. (1990), "Using Yourdon and Z to specify computer security: a case study", Technical Report, IES4/90, Leeds Metropolitan University.

Semmens,L.T. & Allen,P.M., (1991a), Using Yourdon and Z: an approach to formal specification, in "Z User Workshop, Proceedings of the Fifth Annual Z User Meeting, Oxford 1990", J. Nicholls, ed., Springer-Verlag.

Semmens,L.T. & Allen,P.M., (1991b), "A formal definition of the Yourdon essential model", Technical Report, IES1/91, Leeds Metropolitan University.

Semmens,L.T., France,R.B. & Docker,T.W.G., (1992), Integrated structured analysis and formal specification techniques, *The Computer Journal*, Vol 35, No 6 pp 600-610.

Shlaer,S. & Mellor,S.J., (1988), Object-Oriented Systems Analysis: Modeling the World in Data, Yourdon Press : Prentice Hall, Englewood Cliffs, NJ.

Spivey,J.M., (1988) Understanding Z: A Specification Language and its Formal Semantics, Cambridge University Press.

Spivey,J.M.,(1992), The Z Notation: A Reference Manual (2nd edition), Prentice-Hall.

SSADM Version 4 Reference Manual, (1990), NCC/Blackwell, Oxford.

Yourdon,E, (1989) "ModernStructured Analysis", Prentice Hall, Englewood Cliffs, New Jersey

QUELLE: A Knowledge - Based Approach to Systems Development

Vlad Wietrzyk

University of Technology Sydney, Computer Science, Po Box 123 Broadway
NSW 2007 Australia. **vlad@socs.uts.edu.au**

Abstract

One major problem in traditional analysis, design and implementation is the amount of work needed when going from one model to another. The same problem is present in the methods that introduce object-orientation into design and implementation only [Booch, 1986]. The QUELLE project builds on the object-oriented system development paradigm and extends this work in two directions. The first direction is concerned with the utilization of a commercial ODBMS as the underlying data management mechanism. The second direction is concerned with enhancing the paradigm with the explicit modelling of a problem situation providing decision-supporting environment at the application level.

1. INTRODUCTION

A major shortcoming of current practices is that *the modelling of organisational policy within an information system is completely defined only at a low level of abstraction*, namely programming code.

The result is that whereas end users perceive and often define a business system in terms of policies or rules, such a view is not directly visible in the derived system specification. Balzer [1983] has argued for a new approach which recognises the need of addressing system evolution at a high level of abstraction.

Output of requirements engineering should be a formal, complete, precisely defined problem specification, from which during program development code is derived manually or generated automatically. *Automating a process implies taking advantage of machine capabilities to improve upon the original process.*

To achieve automation it is necessary to view the tasks and techniques as parts of a single coherent development process. Support for them must therefore be designed to provide an integrated whole. When this is achieved, the user finds that the automated methodology can then be used in a very free manner.

2. VIEWPOINTS

An information system is a formal mechanism that supports the communication of information among users. The development of information systems is usually made in an organizational context in which several people co-operate towarts the goals of the organization. QUELLE methodology is based on several assumptions which recognise this:

- Each user must be able to construct his own submodels, according to his views of the world.
- Communication between users requires that at least some of the concepts are shared at least partially by the involved users see Figure 1.

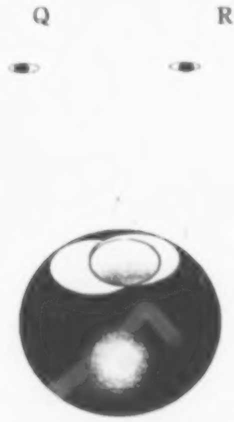


Figure 1 - Users views and domains of interest.

- Users have their own concepts which they use to comprehend their environment and to interact with it.
- A state of continuous change of the environment effects the organization and the working conditions of users.
- The user's change in view is directly reflected by new submodel, discribing part of the world he is interested in, the one - that he needs to carry out his work.
- A user should not be forced to accept the submodel of another user - but the system should enable and help the user to change his reference model.

These assumptions reveal the basic problem in the design of information systems. The knowledge about the universe of discourse must be acquired from users and made shareable within the user community by developing common conceptual model of the community. This empirical model should be object-oriented with specification in a declarative as well as in a procedural mode. Through an analysis of scenarios, arising from design considerations or from a sensitivity analysis of the descriptive model, one looks for a satisfying solution. Subsequently this solution is to be constructed and implemented.

3. ENTERPRISE VIEWS

An enterprise view covers the enterprise objectives of an information system. These include:

- The *Function View* is a representation of the enterprise operation in terms of a structured hierarchy of business processes. Figure 2 depicts the process context level.
- The *Information View* gathers all information defined and contained in the enterprise. The information is structured in information classes and enterprise information objects.
- The *Resource View* contains all the information about the enterprise's resources.
- The *Organization View* contains all relevant information about the responsibilities within the enterprise, and allows for structuring the different responsibilities in the enterprise for function, information and resources.

The enterprise views will be generated one after the other, and this process which is called stepwise generation, will be supported by a set of programs guiding the enterprise designer.

What is needed are software design tools that explicitly model domain knowledge and facilitate reuse of this knowledge by - delivering context dependend knowledge, - consequent design modification will not introduce errors, - requests from users consist of what has to be done by the computer, but not necessarily include a specification of how this has to be done.

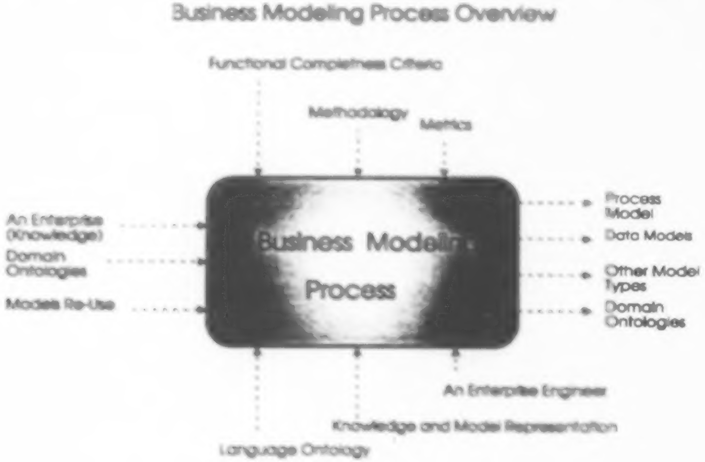

Business Modeling Process Overview
Functional Completness Criteria
Methodology
Metrics
An Enterprise (Knowledge)
Domain Ontologies
Models Re-Use
Business Modeling Process
Process Model
Data Models
Other Model Types
Domain Ontologies
An Enterprise Engineer
Knowledge and Model Representation
Language Ontology

Figure 2 - Process Model

4. DESIGN RATIONALES

Recent studies have confirmed that capturing design rationale is especially important in large-scale development efforts for the following reasons:

- In large projects, over time and with changing development groups, the rationales and context for key design decisions are often confused and even lost. This is largely a result of the inadequacies of current documentation practices in capturing the knowledge about design rationales and decisions.
- Large projects involve multiperson teams. As it is difficult to maintain multiple channels in the absence of mechanisms for representing design decisions, project teams often rely on one or more team members for maintaining communication and coordination.
- Critical errors are commonly made in the formulation and resolution of design decisions, and these are often unnoticed, since they are not recorded.
- Different project teams are involved in different phases of systems development. In the absence of knowledge about design rationales, key design decisions are often misunderstood and misinterpreted.

The goal of this research is the development of a generic model for capturing and representing process knowledge (design rationales and their linkage to design artifacts) across various phases of systems development. One of the aims is to develop a conceptual model to characterize the task of requirements engineering.

5. FLEXIBLE SYSTEM

At a high level of abstraction, a flexible system is a system that can make reasonable progress on any problem instance it is given.
If additional knowledge is required to make progress, then the system should work to acquire that knowledge. To be more specific we consider a flexible system to be a problem-solving system that:

- Allows easy modification of its knowledge through incorporation of a new knowledge or changes to existing knowledge.
- Can engage in complex problem solving about the potential actions that can be used to solve a problem.

A flexible system has the potential to be robust and adaptive - given appropriate knowledge.

6. WHY ARE FLEXIBLE SYSTEMS NEEDED IN INFORMATION SYSTEMS TECHNOLOGY?

There are at least two reasons. First, general intelligence is not defined as the ability to solve a single, bounded problem. An Adaptable Information System must be ready to solve problem thrown at it over the course of its existence. An important defining feature of an adaptable intelligent Information System is its ability to dynamically adapt to solve the problem at hand. If we want to build adaptable intelligent Information Systems then we should expect no less of them.

Second, most problems being tackled when designing Information Systems are so ill-defined that the input and output for a system cannot be precisely specified at design time. Even if it could be precisely specified, the specification might change because:

- knowledge about the task or domain might grow or change;
- the task that the system is designed to do might change (thus requiring a different specification);
- the knowledge available to solve the task may be different than what was assumed when the system was first built [Wietrzyk, 1994].

Thus, the ill-defined nature of most information processing problems combined with the dynamic nature of task environments demand systems that can adapt to new situations, the system must be capable of supporting change.

7. QUELLE: INTEGRATED KNOWLEDGE ASSISTED SOFTWARE DEVELOPMENT ENVIRONMENT

The architecture of QUELLE is an open architecture that is designed for the integration of current and future execution environments without any consequences to users.

Therefore, QUELLE can be seen as a general execution environment that offers the ability to use special execution systems for those tasks where they are most efficient, without forcing the user to face up to technical details.

QUELLE supports the construction of models according to an object-oriented methodology which divides the system in a hierarchy of objects to improve its comprehensibility and to simplify its modification and reuse of system elements. Figure 3 - defines basic object concept as independent computational unit, able to respond autonomously to messages directed to it. It can be also added that the object oriented programming was the first attempt to introduce an anthropomorphic view of computation because objects may be, for design purposes, considered similar to persons with internal data and message language for accepting communications with other objects. This fact provides a source of analogies that may guide the programmer intuition in the design process.

287

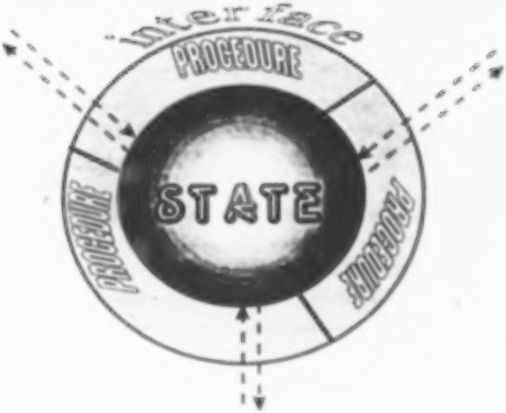


Figure 3 - Basic Object Concept

However there is still a path to be defined from the conceptual understanding of the problem in the programmers mind to the object based or logic programming based computable model. The nature of the knowledge about the problem must be analyzed to define this path.

The approach demonstrated in this paper may help to meet this goal as will be shown in the next paragraph. Figure 4 depicts the basic components of our prototypical system for knowledge discovery in the repository of information.

At the core of the system is the discovery method, which computes and evaluates patterns on their way to becoming knowledge. The inputs to the discovery method include raw data from the database, information from the data dictionary [Wietrzyk 94], additional domain knowledge, and a set of user-defined biases that provide high-level focus. Because this model represents a prototypical system, a discovery system need not include all these aspects.

To solve problems specified by users QUELLE provides the following features:

- the discovery method, which computes and evaluates patterns on their way to becoming knowledge.
- structure and behaviour preserving morphisms from model theory can connect models at different levels of abstraction so that they can be developed to be consistent with each other and can be consistently modified.
- the transformation of an externally represented problem, into a formalized and system-handable internal representation - Figure 5
- the presentation of the internally represented output of the executed method to the user by using visual output.

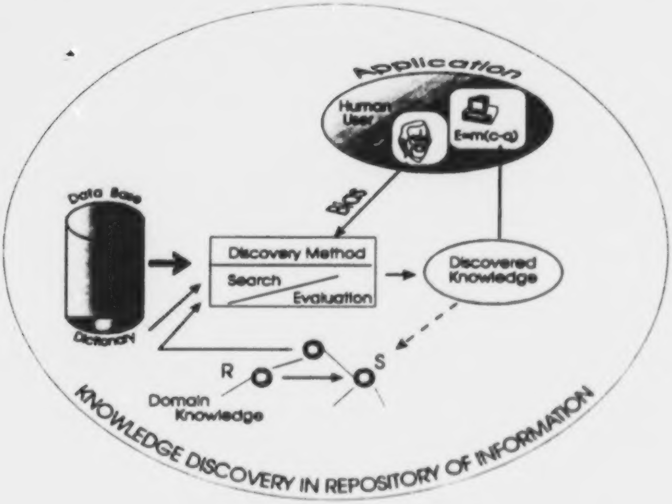

Application
Human User
E=m(c-a)
Bias
Data Base
Dictionary
Discovery Method
Search
Evaluation
Discovered Knowledge
R
S
Domain Knowledge
KNOWLEDGE DISCOVERY IN REPOSITORY OF INFORMATION

Figure 4 - A Framework for Knowledge Discovery in Repository of Information.

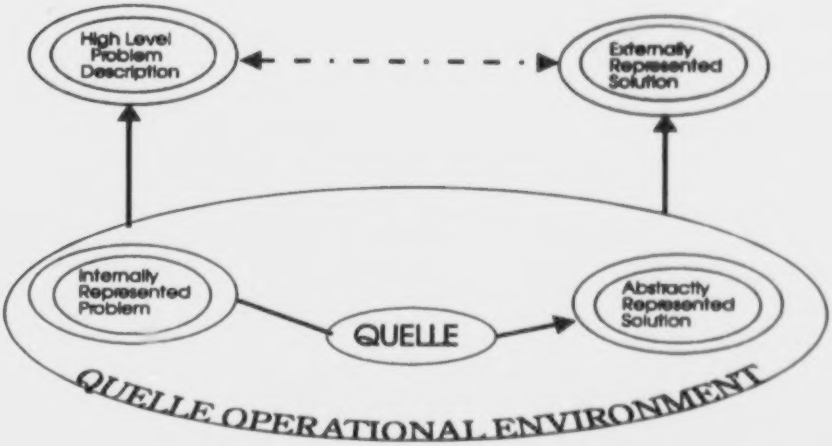

High Level Problem Description
Externally Represented Solution
Internally Represented Problem
QUELLE
Abstractly Represented Solution
QUELLE OPERATIONAL ENVIRONMENT

Figure 5 - The Operational Model of Application.

The use of executable methods normally presumes specific knowledge about existence, location, and handling of the respective programs. Future work in the QUELLE project will concern the integration of an object oriented database like GemStone, VERSANT, etc.. Using an object oriented database to store object persistently allows QUELLE to use the

EST COPY AVAILABLE

features of those systems. The distributed architectures that are offered by most of those database systems support persistent storage of objects that exist in a distributed environment.

8. MODEL BASE DEVELOPMENT

Autonomous component models for operation, planning, decision-making have been developed in their respective fields so they share many commonalities as well as differences in assumptions. In an integrated system, cooperation beetwen them is based on overlaping common functions.

The symbolic level model specification may be formulated based on the format of Figure 7, where knowledge components and inference procedures are displayed.

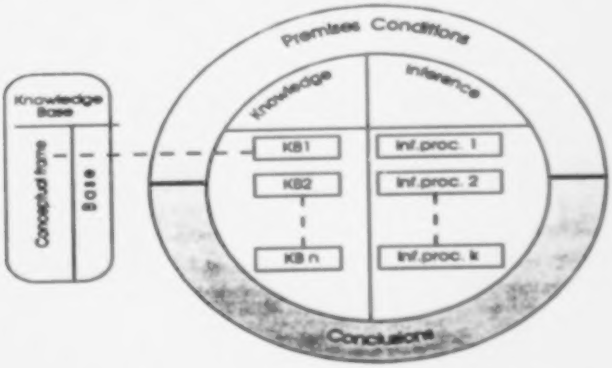


Figure 7 - The general structure of a knowledge object - unit specification.

A knowledge unit specification may be implemented by an object generated at the time of creation. The object will have slots designed for formulation of the conceptual framework, knowledge base and inference procedures. Such type of objects may be considered as the computable compilation of a knowledge unit formulation.

This type of approach may be named knowledge oriented design of Information Systems and it could be considered as a possible new step to follow the object oriented paradigm in the search of antropomorphic models of computation.

From this perspective, the conventional software development or knowledge based development are two successive levels in the application design process but not two alternative incompatible approaches.

9. AUTOMATED DECISION PROCESSES

In many cases, the only purpose of an enterprise activity is to derive information or to make decisions based on information. The considerations and rules for the derivations and decisions can often be fully formalized and the enterprise activities automated, that is, executed by processes in the information system.

Figure 8 depicts a schematic architecture for a decision supporting subsystem based on dynamic, knowledge-based construction of decision models - the decision being maid, relates to the hotel environment, the Front Desk is making a reservation for a guest. If the check on the availability of the room can be fully formalized, then human approval can be replaced by a decision supporting action of the information system. To do this , the required information, including the formal derivation or decision rules, must be available to the automated system. These rules do not describe how a decision is taken or how the information is derived. The rules describe the interrelationships between the availability of rooms and the wishes of the guest. The are rules about situations and behaviour of the involved universe of discourse.

The principles shown in Figure 8 ilustrates the basic form of use of an Information System. In practice, more complex combinations of these forms will occur.

A model construction process is a query-driven stimulated by a request for information about a particular proposition. Query-driven system rely on the user to identify specific events of interest. The advanced functionality of QUELLE must also deal with various types of non-monotonicity. One can envisage a system that selectively modifies a previously constructed model in response to changes in the knowledge base. This indicates the direction of future research on QUELLE integrated subsystems.

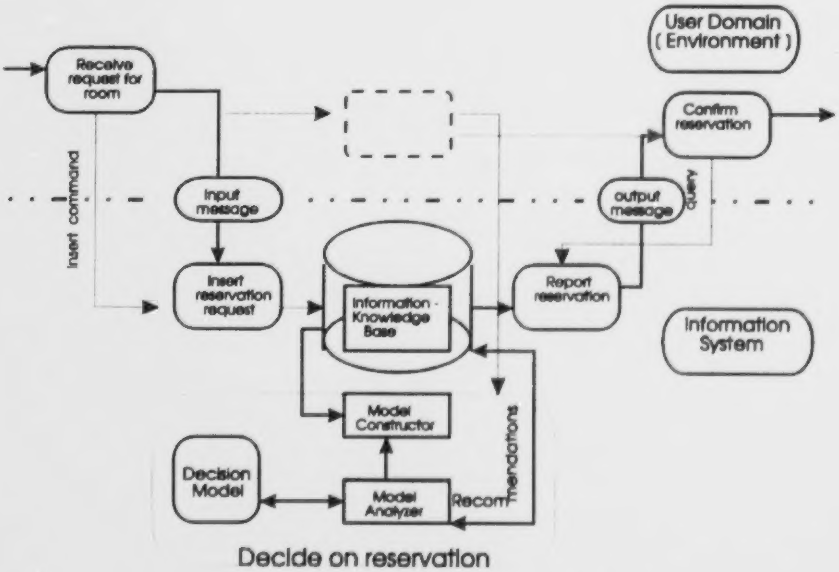

User Domain (Environment)
Receive request for room
Confirm reservation
insert command
Input message
output message
query
Insert reservation request
Information - Knowledge Base
Report reservation
Information System
Model Constructor
Decision Model
Model Analyzer
Recommendations
Decide on reservation

Figure 8 - Automated decision action in the information system.

10. GENERIC ARCHITECTURE

We can associate with each generic problem a set - preferable one, generic architectures (metaarchitecture), which can be used to create a system for solving instances of the generic problem. The blackboard model is one vehicle for implementing opportunistic problem solving. Figure 9 shows the overall concept. Development of Information Systems, which include decision support sub-system, poses a number of requirements including:

- A need for an opportunistic problem-solving methodology ("fusion" of expertise).
- A need to reason with incomplete and possibly inconsistent information.
- A need for extensive communication between heterogeneous representations and solution approaches.
- A need to reason on multiple levels of abstraction.

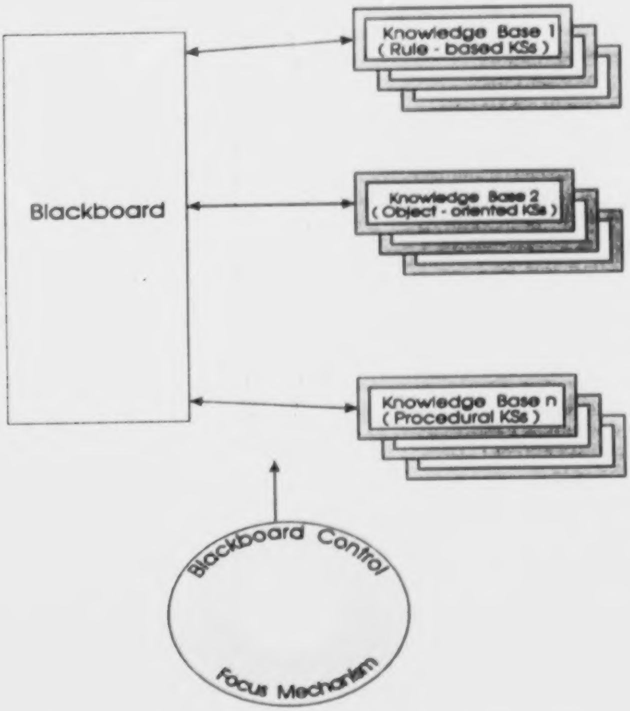

Blackboard
Knowledge Base 1
(Rule - based KSs)
Knowledge Base 2
(Object - oriented KSs)
Knowledge Base n
(Procedural KSs)
Blackboard Control
Focus Mechanism

Figure 9 - Blackboard model for opportunistic problem solving.

A blackboard approach provides environment, fulfilling all of the above needs. We define a strategic model which uses reflection as part of its control framework. A strategy is a process of building up units of knowledge sources. We use Newell's control model to give structure. The control structures are classed as unit and part-of-unit processes. The model of the software system representing an instant of a solution defined on the generic problem space should be detailed enough to enable the automatic code generation in the framework of the operational software life cycle. Software maintenance may then be performed on the model rather than on code.

11. USING THE SYSTEM FOR THE PRODUCTION OF SOFTWARE

The model of expertise finally includes all functional requirements of the desired system. For the realization of the final system additional requirements have to be considered which are still independent of the final implementation of the system. With the goal to create conceptual model amenable to formal verification, a specification technique, based on the conceptual task modelling, is introduced. The technique is based on and defined in terms of predicate/transition nets. In this paper we are not concerned with computational power of the conceptual task model, but we can prove that in the conceptual task model one can simulate any arbitrary Turing machine and by doing this we can prove in fact that the conceptual task model can compute any computable function.

Because of the limited space, the function of all QUELLE components, their mode of operation, and cooperation between them is discussed in the following section in an abbreviated form.

We assume that a generic problem that abstracts the *target environment* problem specification and a generic architecture that abstracts a class of blackboard-based architectures have already been encoded in QUELLE. This software development system starts from the formation of informal requiremants by the clients according to the problem domain as an instance of a known generic problem. The developers, henceforth, do an initial analysis on this informal requirements, and then build a conceptual model. Once a syntactically correct specification is obtained, this specification can be subject to various analyzers which attempt to verify above mentioned software propeties and help in determining the correctness of the specified system.

The dialog-user - interface of QUELLE allows users to edit requests intended to be performed by the QUELLE system. A request is then transformed by the dialog-user-interface into an internally represented problem. An internal representation needs to be validated, i.e., it needs to be checked whether the domain was indeed represented in the internal representation.

If an internal representation is valid with respect to a problem domain, the domain model is isomorphic to a model (an internal representation) of the problem.

To determine whether the internal representation of the problem is isomorphic to the domain model involves testing whether the interpretations of all theorems of the internal representation can be mapped onto facts contained in the domain model. Comparing theorems of an internal representation to the domain allows detecting that the internal representation fails to capture the problem domain adequately before this shows up in the programming and testing phases. The formalism and internal complexity involved in this phase of the development is not discussed in this paper.

12. CONCLUSION AND DIRECTIONS OF FUTURE WORK

In this paper we provided an overview of basic principles and methods of the QUELLE approach.

The architecture of QUELLE features a model base at the center of its planning, operation, decision-making strategies. In this way, it integrates AI symbolic models and control dynamic models into a coherent system.

More research is required to ensure that models in the hierarchy have valid abstraction relations to each other. Additionally, although QUELLE employes a reasonable languages for world modelling and conceptual design, methodologies and development assistants for working with these languages, for capturing requirements in a goal-oriented manner and for integrating views - are still unadequate and further research is necessary.

Obviously, Quelle does currently not cover some relevant areas. Maybe the most important deficiency is the lack of providing means for handling user interface aspects of the Information System being developed in view of capturing domain knowledge related to the universe of discourse.

13. REFERENCES

Balzer, R., (1983), " Software Technology in the 1990's: Using a New Paradigm", Computer, November 1983, pp. 39-45.

Booch, G., (1986), " Object-Oriented Development", IEEE Trans. Software Eng., vol. SE-12, no. 2, p.211, Feb. 1986.

Wietrzyk, V., (1994), "Development of Adaptable Information Systems in Tourism", In Proc. of International Conference on Information and Communication Technologies, Innsbruck, Austria, 1994, Springer-Verlag, 1994.

Organizational Aspects

KEY ISSUES IN INFORMATION SYSTEMS MANAGEMENT. THE CASE OF POLAND

Stanisław Wrycza and Tomasz Plata-Przechlewski

University of Gdańsk, Department of Information Systems, ul. Armii Krajowej 119/121, 81–824 Sopot, POLAND

Abstract

Several research groups have taken up the investigation of Key Issues in Information Systems Management in different countries. The economic changes in Central Europe have created the possibility of accomplishment of the comparable survey in Poland. As a matter of fact, both the current economy and the access to the most up-to-date information technology are becoming similar to those in the developed countries. The selected research method and profile of survey participants are described in the second part of the paper. In the third, main part, the ranking in Information System Management issues in Poland are identified. In the last part Information Systems (IS) Management issues are compared with results of similar studies form the USA and Slovenia.

1. INTRODUCTION

Inspired by the investigation carried out in the USA we have taken up at the University of Gdańsk, Poland the research of key issues in Information System (IS) Management (Niederman et al., (1990), Niederman et al., (1991). The similar surveys were accomplished also in Germany, Slovenia and Taiwan. We assume that results received in Poland are characteristic or typical for the other countries which transform their economies from central planned to market oriented.

It is difficult to compare directly the results of Polish and American (repeated every 3 years since 1980) surveys. First and foremost, it must be taken into account that the access of Polish business organizations and institutions to information technology (both hardware and software) was very limited. The internal obstacles resulted from limited economic resources. On the other hand, the external restrictions have not allowed to keep the pace with the most up-to-date Information Technology (IT). Therefore, in the 80s the field was dominated by personnal computers ("PC monoculture"). In the last three years the situation has been changing quickly. The growing number of professionals has been acquainted with modern concepts, methods and software like information planning, software engineering, CASE (*Com-*

puter Aided Software Engineering) technology, executive and decision support systems, object-oriented systems and many others.

2. METHOD AND SURVEY PARTICIPANTS

The survey was based on the questionnaire consisting of two parts. First one, which can be called identification part, includes several questions which allow to asses the respondents size, branch of economic activity, number of Electronic Data Processing (EDP) department staff, hardware and software equipment, respondents market strategy versus information systems strategy, etc. Second part of the questionnaire consists of 25 key information systems development and implementation issues. Participants of the survey were asked to rate (rather than rank) each issue on 10-point scale where 10 indicated their highest priority issue(s) while 1 indicated their lowest priority issue(s).

During the survey 500 questionnaires were distributed among participants of conferences, seminars and fairs devoted to modern information techniques. Usable responds were received from 107 respondents all over the country, yielding a rate of response of ca 20%. Average number of total employment in the sample is 1,125 persons (median 255, mode 80).

3. KEY ISSUES IN INFORMATION SYSTEMS MANAGEMENT

Summary results from the second part of questionnaire are presented in table 1. The most important issue among 25 presented to respondents appeared to be *Making Effective Use of the Data Resources* (question no 3). In respondent's opinion these resources are still not as highly appreciated as other factors of production (i.e. labour or capital) in theirs organizations. Ranked second in importance is *Facilitating/Managing Decision and Executive Support Systems* (question no 21) and third *Developing an Information Architecture* (1). It seems that Polish IS executives pay the biggest attention to strategic issues of IS development in business organizations. This conclusion is supported by the results of the first part of the questionnaire.

The lowest ranks were obtained for: *Planning and Using CASE Technology* (18), *Planning and Using Image Technology* (23) and *Developing and Managing Distributed Systems* (17). Therefore, technological issues were ranked low on average. The reason for this negative phenomenon may be the insufficient knowledge of Polish EDP department staff as far as modern IS technologies are concerned.

In the last column of table 1 standard deviation of individual rates are presented for each question. It measures degree of consistency or inconsistency in presented opinions. It appeared that opinions about issues 3 and 21 were the most consistent (lowest values of standard deviation) — all respondents highly rate them, while opinions about issues no 18, 24 (*Moving Toward Global Systems*), 23, 17 differ between respondents.

A major feature of the survey analysis is the search for variation in responses between subgroups of a sample. In this part of our paper the attempts of associate variations with differences in some economic variable are made. The extensiveness

Table 1. Top Key Issues in Information Systems Management

Rank	Description of the issue	No	Rating	st.dev.*
1	Effective Use of the Data Resources	3	8.183	1.894
2	Executive/Decision Support	21	7.433	2.413
3	Developing an Information Architecture	1	7.396	2.669
4	Telecommunications Systems	9	7.250	2.779
5	Software Development	10	7.204	2.545
6	Security and Control	15	7.114	2.843
7	IS Human Resources	4	6.912	2.758
8	Facilitating Organizational Learning	7	6.931	2.475
9	Electronic Data Interchange	11	6.933	2.880
10	Technology Islands	14	6.942	2.425
11	IS Role and Contribution	8	6.792	2.609
12	Disaster Recovery	22	6.615	2.947
13	Strategic Planning	2	6.410	2.892
14	Building a Responsive IT infrastructure	19	6.109	2.983
15	Competitive Advantage	5	6.058	3.193
16	IS Assets Accounting	25	5.990	2.809
17	IS Organizational Alignment	6	5.822	3.057
18	Applications Portfolio	12	5.716	2.596
19	Organizational Structure	20	5.798	3.117
20	End User Computing	16	5.423	2.384
21	Global Systems	24	5.136	3.224
22	IS Effectiveness Measurement	13	5.157	2.504
23	Distributed Systems	17	4.901	3.161
24	CASE Technology	18	4.306	3.254
25	Image Technology	23	4.279	3.185

* standard deviation of individual rates.

of division is limited by the sample size, but nevertheless some results seem worthy to be mentioned.

First, the relation between the size of an institution and estimation of key issues were analyzed. The surveyed companies were divided into the following groups:

- small enterprises — not more than 100 employees,
- medium firms — 101–1000 employees,
- big companies — more than 1000 employees.

It appeared that respondents working in bigger institutions on the average rate higher issues like: *Improving IS Strategic Planning* (question no 2), *Making Effective Use of the Data Resources* (3), *Specifying, Recruiting, and Developing Human Resources for IS* (4), *Facilitating/Managing Decision and Executive Support Systems* (21), *Facilitating Organizational Learning and Use of IS Technologies* (7), *Increasing Understanding of the Role and Contribution of IS* (8), *Planning and Effective Management of the Applications Portfolio* (12), *Measuring IS Effectiveness and Productivity* (13), *Planning and Using CASE Technology* (18), *Using IS to Influence Organizational Structure* (20), than those from smaller firms. Answers to issues *Enabling Electronic Data Interchange and Multi-Vendor Integration* (11) and *Image Technology* (23) show reverse relation. It seems that smaller firms seek their market chance in modern information technologies as *Image Technology* and *Electronic Data Interchange*. Five top issues for each group of firms are presented in table 2.

Table 2. Top Key Issues in Information Systems Management by Total employment

Rank	Description of the issue	Rating	st.dev.
	1–100 Employees		
1	Data Resource	7.9722	1.9926
2	Information Architecture	7.4857	2.7586
3	Software Development	7.4000	2.4758
4	Electronic Data Interchange	7.0556	3.1071
5	Security and Control	6.9444	3.1254
	101–1000 Employees		
1	Data Resource	8.1739	1.8655
2	Executive/Decision Support	7.6522	2.5492
3	Telecommunications Systems	7.6304	2.4162
4	Technology Islands	7.3696	2.2643
5	Security and Control	7.2979	2.7969
	1001 and more Employees		
1	Data Resource	8.5455	1.8186
2	Information Architecture	8.2273	2.2239
3	Executive/Decision Support	7.9545	2.1487
4	IS Human Resources	7.7273	2.2716
5	Organizational Structure	7.6364	1.9651

An Analysis by branch of *economic activities* shows considerable differences in IS managers across certain surveyed groups. Companies in electronic industry tends to rate higher (on average) all 25 issues than companies from any other group. On the other hand, respondents in tourist services rate them low. There are also significant differences in answers to individual questions. IS executives in transport and communication, banking and tourist sectors attach more importance to *Planning and Implementing a Telecommunications Systems*, which may be connected with the problems they encountered in their everyday work. Due to problem of infallibility of economic and technological processes banks and respondents in industry pay high attention to *Improving the Quality of Software Development*. *Improving Information Security and Control* is an issue highly rated by respondents in the administration.

An Analysis by total employment in EDP department shows significant differences between surveyed institutions. Exact figures are shown in table 3.

For big EDP departments (employing more than 50 persons), the most important appear to be following issues: *Developing an Information Architecture*, *Improving IS Strategic Planning*, *Aligning the IS Organization with that of the Enterprise* and *Using IS to influence Organizational Structure* as well as *Facilitating/Managing Decision*

Table 3. Top Key Issues in Information Systems Management by Total employment of EDP department

Rank	Description of the issue	Rating	st.dev.
	1–3 Employees		
1	Developing an Information Architecture	8.1186	2.0434
2	Global Systems	7.6780	2.4029
3	Electronic Data Interchange	7.3729	2.8460
4	Building a Responsive IT infrastructure	7.3729	2.2507
5	Effective Use of the Data Resources	7.2500	2.7253
	4–10 Employees		
1	Developing an Information Architecture	8.7368	1.4080
2	IS Role and Contribution	7.4211	2.7145
3	Facilitating Organizational Learning	7.1579	2.8139
4	Telecommunications Systems	7.1111	2.3235
5	Competitive Advantage	7.0000	3.3500
	11–50 Employees		
1	Technology Islands	8.2917	1.8992
2	Effective Use of the Data Resources	8.1667	2.2778
3	Executive/Decision Support	8.0833	1.9982
4	Developing an Information Architecture	7.9583	1.8528
5	Global Systems	7.5000	2.2263
	50 and more Employees		
1	End User Computing	9.0000	1.4142
2	Security and Control	9.0000	1.4142
3	Effective Use of the Data Resources	9.0000	1.4142
4	Executive/Decision Support	8.0000	2.8284
5	Global Systems	8.0000	0.0000

and Executive Support Systems. It seems that EDP department staff from those big units pay much attention to strategic issues; less important for them are: *Specifying, Recruiting, and Developing Human Resources for IS*, *Using Informations Systems for Competitive Advantage*, *Planning and Implementing a Telecommunications Systems*, *Electronic Data Interchange*, *Measuring IS Effectiveness and Productivity*, *End User Computing* and *Building a Responsive IT Infrastructure*. Low rating of some issues may indicate lack of dynamics in transformation to market demands.

Small EDP departments highly rates *Facilitating Organizational Learning and Use of IS Technologies*, *Measuring IS Effectiveness and Productivity*, *Integrating Data Processing, Office Automation, Telecommunication, and Factory Automation* and *Accounting for the Asset Value of Information and Software*. It responds to specifity of small firms offering their systems to users and forced by the market necessity of financial result analysis. Global issues or those which are connected with implementing modern information technology are rated rather low by this group of respondents.

Mid-size EDP departments attach much importance to *Facilitating and Managing End-User Computing*, and *Planning and Management of the Applications Portfolio*. There are no significant differences in rating the following issues: *Improving Informa-*

Table 4. Top Key Issues in Information Systems Management by strategies of implementing new IS

Rank	No	Issue	Average	Variance	n^*
		Purchasing and developing			
1	03	Data Resource	8.1071	1.8457	56
2	10	Software Development	7.8214	2.3441	56
3	21	Executive/Decision Support	7.5357	2.2317	56
4	01	Information Architecture	7.4286	2.3422	56
5	09	Telecommunications Systems	7.3393	2.5956	56
6	04	IS Human Resources	7.2500	2.5100	56
7	07	Organizational Learning	7.0909	2.5113	55
8	02	Strategic Planning	7.0000	2.5963	55
9	22	Disaster Recovery	6.9107	2.9743	56
10	15	Security and Control	6.8750	2.9730	56
		Exclusivly purchasing			
1	03	Data Resource	8.4000	1.9189	40
2	21	Executive/Decision Support	7.4250	2.5206	40
3	15	Security and Control	7.1951	2.7948	41
4	01	Information Architecture	7.1081	3.0982	37
5	14	Technology Islands	7.0500	2.5914	40
6	09	Telecommunications Systems	7.0500	3.0209	40
7	07	Organizational Learning	6.8974	2.3260	39
8	11	Electronic Data Interchange	6.8293	3.0323	41
9	08	IS Role and Contribution	6.4737	2.5757	38
10	22	Disaster Recovery	6.4500	2.9348	40

* number of respondents.

tion Security and Control, *Developing and Managing Distributed Systems* and *Planning and Using Image Technology*.

Analyzing approaches exploited by respondent in systems development the companies were divided into two groups: those who only purchase and adapt software and those who develop IS as well as purchase it (combined strategy). The latter group on the average rates higher all key issues except *Making Effective Use of Data Resources*. *Software Development* are ranked higher by those who use combined strategy in implementing new systems while *Improvement of Information Security and Control* by the opposite group. This results seems to be quite natural comparing to strategies realized by respondents from respective groups (compare table 4).

4. INTERNATIONAL COMPARISON

Comparison with results from similar surveys in the USA and Slovenia is presented in table 5. In the table top ten and bottom six issues are presented. In Slovenian survey two issues were generated by respondents during the stage Delphi study (see Dekleva and Zupančič (1992)), so comparing to original US survey some issues are missing.

Most of the top 10 issues in our study fall into issues generated by the surveys in the US and Slovenia. We can observe considerable difference as far as *Facilitating/Managing Decision and Executive Support Systems* (21), *Improving Information Security and Control* (15), *Integrating Data Processing, Office Automation, Telecommunication, and Factory Automation* (14), which are ranked much higher by Polish IS executives than their US counterparts. On the other hand *Develpoing and Managing Distributed Systems* (17) and *Planning and Using CASE Technology* (18) shows reverse relation. This confirms that the key issues in IS management is Poland differ from those in the USA (and probably all Western countries) and are closer to those from Slovenia.

Table 5. Key Issues in Information Systems Management. International Comparison

Rank	Description of the issue	USA	Slovenia
1	Effective Use of the Data Resources	2	–
2	Executive/Decision Support	17	8
3	Developing an Information Architecture	1	7
4	Telecommunications Systems	10	9
5	Software Development	9	–
6	Security and Control	19	–
7	IS Human Resources	4	2
8	Facilitating Organizational Learning	5	1
9	Electronic Data Interchange	12	12
10	Technology Islands	22	–
—	—	—	—
20	End User Computing	18	–
21	Global Systems	22	–
22	IS Effectiveness Measurement	16	25
23	Distributed Systems	12	21
24	CASE Technology	12	11
25	Image Technology	24	–

5. SUMMARY

The survey of key issues in Information Systems Management was carried for the first time in Poland. It appeared that there is no custom to participate in such surveys—the professionals were reluctant to fill the questionnaires. The post form of survey failed completely and the work of *interviewers* was necessary. The final rate of return was ca 20%.

In general, the priority given to *strategic issues* of Information Systems development and maintenance is rather high. Over 60% of IS departments in questioned organizations is subordinated directly to the president. Most of organizations confirm the necessity of Information Plan Development, while this plan exists in 32.4% of firms and institutions. Also strategic issues like: *Effective Use of Data Resources*, *Executive Decision Support* and *Developing an Information Architecture* received the highest ranks.

The strategic thinking in IS department however, can not be supported by practice. The field of Information Systems is dominated by purchases and adaptation of application packages (83% of examined firms). The next on the list is prototyping, while structured methodology, automated approach or system development life cycle paradigm received low ranks.

The investigation establishes a basis for *repeating* such surveys and comparing them periodically in the future.

REFERENCES

Dekleva S, Zupančič J., (1992), "An empirical investigation of key issues in information system management", *in*: "Information Systems Developers Workbench", Wrycza S. (ed), Gdańsk University Press, Sopot.

Dexter A.S., Janson M.A., Kiudorf E., Laast-Laas J., (1993), "Key information technology issues in Estonia", *Journal of Strategic Information Systems*, vol. 2, no. 2, pp. 139–152.

Doukidis G.I., Smithson S., Naoum G., (1992), "Information management in Greece: uses and perceptions", *Journal of Strategic Information Systems*, vol. 1, no. 2, pp. 63–75.

Niederman F., Brancheau J.C., Wetherbe J.C., (1990), "Information systems management issues in the 1990s", Management Information Systems Research Center, MISRC-WP-91-08, University of Minnesota, Minneapolis.

Niederman F., Brancheau J.C., Wetherbe J.C., (1991), "Information systems management issues for the 1990s", *MIS Quarterly*, pp. 475–500.

Watson R.T., Brancheau J.C., (1991), "Key issues in information systems management — an international perspective", *Information Management*, vol. 20, pp. 213–223.

Wrycza S. (ed), (1992), "Information Systems Developers Workbench", Preceedings of the 3rd International Conference, Gdańsk University Press, Sopot.

SOME RECENT ORGANISATIONAL AND SOCIAL CHANGES IMPACTING UPON EXTANT INFORMATION SYSTEMS METHODOLOGIES

Stephen K. Probert[1]

[1]Department of Computer Science, Birkbeck College, University of London, Malet Street, London WC1E 7HX, United Kingdom

Abstract

Some important changes in the organisational and social climate in which information systems (IS) are developed are discussed. These changes are relevant in that they constrain the applicability and limit the relevance of many extant IS development methodologies. Specifically, three issues are discussed. Firstly, the impact of the move to a more flexible labour force in organisations; secondly, changes taking place in the *style* of management; thirdly changes brought about in the very nature of social reality by the widespread use of information technology. It is argued that this third phenomenon has philosophical consequences that are critically relevant to the creation of sound IS development methodologies. For all three issues the impact of the phenomenon, in terms of its effect on the applicability and relevance of IS development methodologies, is discussed and conclusions concerning the future relevance of extant methodologies are provided.

1. INTRODUCTION

Important changes are taking place in organisations, particularly in terms of workforce composition and management styles. The evidence used in this paper is from the United Kingdom, where it can be argued that these changes have happened more quickly and affected organisations more deeply than in many other western European countries, and although it is a matter of conjecture as to whether other western European countries will experience these changes to the same extent, there can be little doubt that they will experience similar changes *to some extent*. These changes have happened and are continuing to happen so fast that it is not the fault of the propounders of IS development methodologies that some of their methodologies risk becoming less and less relevant to the organisations in which they are intended to be used (indeed the reverse situation would be highly surprising).

The paper concentrates initially on highlighting some key changes taking place in workforce composition, and the first of these changes (the impact of the move to a more flexible labour force in organisations) can be supported with a plethora of empirical fact. The second of these changes (changes taking place in the *style* of management) is less easy to support with concrete facts, as it arises in part from a series of more-or-less logical conclusions as to the nature of the management role in a turbulent, hyper-competitive, environment. Management theorists such as Tom Peters (1993) argue for the necessity of these new styles of management partly from *contingent* real-world experience of what "modern" managers do and partly from a kind of (historically contingent) *a priori* reasoning about what managers must *necessarily* do in a modern (western) organisation, if it is to be competitive. (The results from this *a priori* reasoning form the basis of hypothetical imperatives - i.e. *prescriptions* - for management action-agendas.) The latter part of the paper focuses on the changes that have taken place in organisations as a result of the widespread use of information technology; here the analysis is supported by reference to some of the arguments put forward by post-structural social theorists.

2. ORGANISATIONAL STRUCTURES AND THE FLEXIBLE LABOUR FORCE

There is a considerable amount of extant literature about "organisations of the future". However, some important challenges to the prescriptions of IS development methodologies may lie in store from what seem, at first sight, to be relatively benign developments. The so-called "flexible labour force" will provide one such challenge. For example, in 1989 the author undertook a number of consultancy assignments in a United Kingdom service organisation with about 20 full-time employees in its head office - but at least double that number of *temporary* employees. According to Charles Handy (1991) such organisations will increase at the expense of organisations with a more traditional mixture of mainly full-time employees, plus a few "temps", for a variety of reasons (e.g. the move from manufacturing industries to service industries in the United Kingdom). The organisation that has been briefly described was quoted on the British Stock Market despite the fact that the accountants, the computer staff, the clerks etc. were mainly "temps". In such environments the scope for user-participation is dramatically reduced, owing to the transitory and itinerant nature of the *majority* of personnel. What is needed in such organisations are information systems that are simple to use and yet highly flexible. Much of the IS literature concerned with improving the practice stresses the need for increased user participation. In service organisations which rely heavily on "flexible" labour such arguments will rapidly diminish in relevance. Handy estimates by that the year 2000 there will be 26 million people in paid employment in the United Kingdom, but only half of these will be in full-time permanent employment. The other half will be in part-time or temporary employment (or self-employment). The salient aspects of the employment situation in the United Kingdom have been recently researched by several academic institutions, and have been summarised recently thus:

> By the year 2000, 1.2m mainly full-time jobs in UK manufacturing and utilities are expected to go. Over the same period nearly 2m jobs in services will be created, but more than half of them will be part-time - a trend underlined by ... [the] announcement from the Burton Group [a clothing retailer] that 1000 full-time jobs were being converted into 3000 part-time ones. (Goodhart, 1993)

Handy does not suggest that the drift away from full-time permanent employment as the norm will "stabilise out" at 50/50 (% full-time/other), so it could be that the drift away from full-time permanent employment will continue well into the next century. Furthermore it can be argued that the "flexible organisation" is, itself, only made practically possible by information technology, so as the adoption of information technology continues, so the possibilities for changing the employment composition from full-time to part-time or temporary workers increases correspondingly.

3. MANAGEMENT AND LEADERSHIP

As the price of hardware has fallen, and software has become easier to use, the agenda for IS problems has increasingly become user-driven, as users in the workplace have increasingly acquired "their own" *personal computers* (the language is significant). Managing this ubiquitous technology is one of the pressing problems facing management (generally) and therefore much of the IS agenda is management-driven; "the problem" is now often re-cast as being one of "information management".

Looking back, over the past twenty or thirty years the shift of emphasis from technological issues to management issues, might (crudely) be explained by an economic relationship between *price* and *performance*. If the processing-power-per-dollar continues to fall rapidly then changes to the IS agenda can be anticipated. (Anecdotal estimates of future price-performance are typically of the following form: that by the end of the millennium the most powerful desktop computers in existence today will be available on a credit-card, they will cost about $10.00 - and they could therefore be thrown away when they fail.) The changes that have occurred in the "IS agenda" could be emphasised by the nature of the (managerial) questions that have been and will be asked, i.e.:

e.g. 1970 'What can we do with computers in our organisation?'
e.g. 1985 'What ought we be doing with the information systems in our organisation?'
e.g. 2000 'What might people want to use information systems to do in our organisation (and can/should people be helped to achieve these aims)?'

The first sort of question needed a technological sort of answer - an example answer might be "payroll". However it is widespread knowledge that, despite the technical attractions of payroll systems, such systems are "socially sensitive" - for example they have work correctly every week (or month)! But for as long as the question asked was 'What can we do with computers in our organisation?' the answers would always be related to those things that appeared to be the most "doable".

The second sort of question needed a managerial sort of answer (usually based on "business need") - such answers are the dominant sorts of answers today, although there is a developing trend for these answers to take a more strategic view of the *possibilities for exploiting* the technology (e.g. Earl, 1989). This indicates a shift from the second sort of question to the third - although the strategic questions are essentially still management questions.

What makes the third sort of question different from the first two sorts is that it presupposes that the employees themselves have a vision of what the organisation is trying to achieve, and that they share the value system embodied in that vision; that is it implies the need for a shift from *management* to *leadership* (see e.g. Handy, 1991: 105-108). But such a change may force a re-think of some other key concepts utilised in IS methodologies; these changes will now be discussed.

4. CONSIDERATIONS FOR IS METHODOLOGIES

IS development methodologies generally assume that some (and - oddly - usually one) *analyst/practitioner/researcher* is going to "do the job", even on a facilitative model. (In philosophical terms this straight-away implies a *subject-centred epistemology* and - correspondingly - a *"problematic" ontology*.) In practical terms this arises because the analyst is conceived as having to make sense - on his or her own - of a difficult or puzzling situation; in exactly the same way as a Newtonian scientist would try to make sense of a puzzling natural phenomenon. The subject-centred model is appropriate to questions of the first two types (above), because in the case of "what can we do with computers" it is clear that this question is highly scientific in nature - and is slightly ambiguous, because whilst one interpretation might be "what can computers do technically?" another would be "can we do the things that we currently do in our organisation better/faster/more accurately with computers?" In either case these are questions that appear to have an answer that can be ascertained by the use of Newtonian-style research strategy. In the case of "What ought we doing with our information systems?" the question is in part technical or scientific but in part normative - it requires that the question "What ought we to be doing in our organisation?" has an answer; which is in part a moral question. However the subject-centred model is not so useful to the third type of question, essentially because the subject-centred model is, in a key sense, a *researcher-based* model. *Infusing people with a vision* is not normally considered to be a research activity, and is arguably not a very well-understood issue *per se*; for the IS methodology propounders, it needs to become one, and it should be recognised that infusing people with a vision is an issue which has a very large ethical component.

To discuss IS methodologies specifically, there exists a tendency to simplistically equate "hard" approaches to IS development with "positivistic" natural science, and to equate "soft" approaches with some other sort(s) of "social" methodology(ies). These arguments are generally presented as being "definitely the case". However, much recent philosophy of science (and sociology of knowledge) argues that *science has no definite method* (e.g. Feyerabend, 1993; Rorty, 1980). Feyerabend's argument can be summed up as follows:

> Philosophers of Science such as Popper or Lakatos have tried to define "The Scientific Method". But Feyerabend has shown that if scientists had followed their methodologies modern science would never have advanced a single step beyond its medieval predecessors. Galileo would have had to abandon Copernican astronomy, Einstein would have given up relativity. This is not to say that scientists never follow any rules or that science is completely irrational and subjective. Although Feyerabend admits his affinity to the irrational, his point has always been more subtle: there are no rules that apply to all science at all times. As science progresses, the criteria by which new theories and observations are judged change along with these theories and observations. (Weber, 1993)

Furthermore, all the homilies about the virtues of "interpretivism" (e.g. Stowell, 1993) tend to ignore recent - and arguably important - developments in philosophy which castigate the value of "subjective immediacy" (the facile tendency to take what people say as face-value evidence for what is the case in society); a tradition which can in fact be traced back to the 1930s (see Adorno, 1973). This (critical) tradition is now more generally associated with Foucault (e.g. 1982) and the "postmodern" philosophers. Again, the IS development literature is impoverished by the over-simplification of some of these key issues. More importantly, placing so much emphasis on "subjective immediacy" - whilst helpful in answering the first two sorts of questions (above) - may be positively dangerous in an environment in which "leadership" is a key concept; there are numerous examples of such dangers embodied in the views of extreme political leaders (and their followers) in the twentieth-century. There is a need to make an ethical judgement here, and the reader is invited to consider whether he or she takes what their leaders say at face-value. Many forms of political extremism exploit the fact that people often believe things and have opinions which, taken at face value, become fodder for cynical manipulation (such as racist views), and to effectively criticise extreme political views one needs an objective standpoint - and, if this is not available, a dialectical position may the best that can be attained.

4. TECHNOLOGY (AND "INFORMATION")

Until recently most IS methodologies have focused on "textual" deliverables, printed reports, on-screen text, etc. Zuboff (1988) uses the phrase "electronic text" to describe the outputs of such systems. This electronic text creates both new problems and new opportunities:

> The computerization of productive and administrative processes in an organization reproduces some of these effects of the written word, but it does a great deal more as well. The technology takes over for a certain amount of human activity, even as it renders that activity in text... activities are made transparent. They are exposed in detail as they are textualized in the conversion to explicit information - that is informating. (Zuboff, 1988, p. 181)

Zuboff's work shows that it is difficult enough to fully appreciate the changes that have *already* taken place as a result of the widespread introduction of computerised IS. The phrase "data manipulation" usually refers to the manipulation of words and numbers - and yet increasingly it may not appear to be words and numbers that are manipulated but rather sounds and images (and possibly even tactile sensations). Virtual reality, electronic whiteboards, etc, may well determine important aspects of the "information systems" of the future. Here some important issues are being overlooked by the propounders of IS development methodologies. In the U.S.A. it has been argued that it was television that had a powerful effect on public opinion concerning the Vietnam war; as Marshall McLuhan put it "the medium is the message". In the past (and the present) "information" was almost always textual; although now it is often graphical. In a future of computer-mediated sounds and images (or even the much-hyped "virtual reality"), it may not be desirable to talk of "information".

One might not wish to talk about the televised images of starving peoples as "information", as the word "information" does not seem to characterise the very "direct" quality of such experiences. The difference between information systems and virtual reality could be described as the difference between the *representation* of some state of affairs and the *mediated perception* of some state of affairs. This difference has methodological implications for "information systems" development, and these will now be explored further. It could be argued that *Human Computer Interaction* writers are concerned with such issues, and that (e.g.) Winograd and Flores (1987) demonstrate similar concerns. However most of the current IS development literature is restricted to purely "grammatocentric" (linguistical and numerical) requirements - the "ordering and gridding" of data into tables etc. As such this literature will be increasingly inadequate - owing to the changing nature of both the technology and the world at large. Coming to terms with these new technologies will soon be an important challenge for the IS development methodology propounders.

Peter Drucker, in a section of his 1989 book entitled "From analysis to perception", puts forward the following argument (which is dialectical in tone) implying that changes in our epistemological assumptions need to be made:

> Technology is not nature, but man. It is not about tools. It is about how man works... But precisely because technology is an extension of man, basic technological changes always both expresses our world view and, in turn changes it. The computer is in one way the ultimate expression of the analytical, the conceptual world view... Since Descartes, the accent has been on the conceptual. Increasingly we will balance the conceptual and the perceptual... It took more than a hundred years after Descartes and his contemporary, Galileo, had laid the foundations for the *science* of the mechanical universe, before Immanuel Kant produced the *metaphysics* that codified the new world view... But contemporary philosophers no longer focus on Kant's concerns... They deal with perception. Thus the shift from the mechanical to the biological universe will eventually require a new philosophical synthesis. (Drucker, 1989, pp. 251-254)

The main examples of social changes provided so far are really symptoms of other, more important changes currently taking place. Moreover it can be argued that the changes that are taking place today have profound implications for the social theories which the IS development community has grown accustomed to, as Mark Poster suggests:

> Social theory arose in a Cartesian culture of distinct objects and subjects, in a dualist metaphysics of extended things and minds. In this theoretical context, the social scientist is constituted as a knowing subject separate from his or her object of study, one who enunciates univocal words to define an objective social field distinct from himself or herself... Today however the representational character of language is especially fragile and problematic. In sphere after sphere of daily life, the relation of word and thing is complicated by the loss of the

> referent. (Poster, 1990, p. 12)

Poster argues that the widespread use of computerised information systems has changed the basic relationships between words and things, and this aspect of his argument goes considerably beyond the argument put forward by Drucker. The essence of the argument is as follows. Given that language can often no longer be assumed to refer to a "thing" - other than a very arbitrary "thing" indeed - there can be no coherent argument for basing epistemology on a (linguistic) representational basis. The whole idea of "models" in the mind of subjects has been supplanted by more transient notions of reality:

> The function of representation comes to grief when words lose their connection with things and come to stand in the place of things, in short when language represents itself. The complex linguistic worlds of the media, the computer and the databases it can access, the surveillance capabilities of the state and the corporation, and finally, the discourses of science, are each realms in which the representational function of language has been placed in question by different communicational patterns each of which shift to the forefront the self-referential aspect of language. (Poster, 1990, pp. 12-13)

Poster concludes not that there is a very peculiar aspect to language in the modern (western) world but that the whole theory of "subjective representational perception" is no longer tenable - and was most likely flawed from the outset. Such changes have a profound effect on the project of a salient and coherent "social epistemology" upon which to base IS development methodologies - primarily owing to the (conceptual) instability of the "action-centred subject":

> ...[C]ontemporary society ought not to be approached through action-centred models... In order to take account the politics of the "other", critical social theory requires an epistemological overhaul in which the figure of the rational subject no longer serves as a ground or frame. (Poster, 1990, p. 17)

As information systems development becomes more complex and invasive of everyday life so an action-centred model of the development process will become less and less relevant. IS methodologies should acknowledge the role that the IS developers play as forces for social reproduction or social change, and furthermore the propounders of IS methodologies should do so as well, and be more explicit about their social/political ambitions. The questions concerning the relevance, appropriateness, and ethicality of any particular IS development methodology cannot be asked in isolation from the questions concerning the assumptions made about social reality. IS development methodology

311

propounders should be sufficiently astute as to avoid attracting obvious criticisms from others with a more critical imagination.

5. CONCLUSIONS

5.1 Organisational Structures

Issues raised by changes in organisational structures, in particular the demise of the full-time core employees, can no longer be comfortably avoided by the IS methodology propounders - especially as IS developers may have been instrumental in their demise.

5.2 Management and Leadership

Understanding the process of infusing people with a vision will be an important issue for IS development in the near future, and that "the vision thing" will play a key role in determining the outcomes of IS developments in organisations. If so a new approach - new to many IS development methodology propounders at any rate - to understanding the relationship between epistemology and ontology will be needed. Consequently, the relationship that will most usefully be considered may be a dialectical one, as this will cater for the analysis of the historical changes impacting on the salient aspects of the IS development environment. Obviously there will be other relevant "conceptual contenders" for such analyses, but more debate about such issues is urgently required. In fact it may prove to be positively dangerous to try to avoid these issues, as there is a growing risk of strong *authoritarianism* emerging as the key ideal of IS development in many organisations - via the *leadership ethic* (coupled with enforced obedience).

5.3 "Information"

The media by which computer-based data is presented is changing to the point where the term `information' itself may already be becoming outmoded by technological developments. Some new concepts are necessary, as the epistemological hegemony of representational perception (on a subject-centred model) in many common approaches to IS development needs to be overhauled. It may be better to accept that one perceives "directly" - but that what one perceives is in a sense often "unreal" (or hyperreal - to use Baudrillard's term) and is therefore capable of critical interpretation.

Until recently, information systems were limited to (what can broadly be described as) *representation*. In all manner of ways, future systems will more-and-more create *reality itself*. The challenges posed by such a state of affairs are immense, as are the opportunities as what can be created can be used for a variety of ends - not all of which are ultimately desirable! Some of the arguments and criticisms put forward herein apply more to some IS methodologies than to others but space has prevented a more detailed analysis of specific IS methodologies, although clearly this would have been a valuable exercise.

REFERENCES

Adorno, T. W., (1973), "The Jargon of Authenticity", (translated by K. Tarnowski and F. Will), Routledge and Kegan Paul, London.

Drucker, P., (1989), "The New Realities", Mandarin, London.

Earl, M. J., (1989), "Management Strategies for Information Technology", Prentice Hall International, Hemel Hempstead.

Feyerabend, P., (1993), "Against Method", 2nd ed., Verso, London.

Foucault, M., (1982), The subject and power, *in*: "Michel Foucault: Beyond Structuralism and Hermeneutics", H. Dreyfus and P. Rabinov, Harvester, Brighton.

Goodhart, D., (1993), Just the job for the future, *Financial Times*, January 8 1993, p. xx.

Handy, C., (1991), "The Age of Unreason", 2nd ed., Business Books, London.

Peters, T., (1993), How well will you manage in 1994?, *in*: "The World in 1994", The Economist Publications, The Economist Newspaper Limited, London.

Poster, M., (1990), "The Mode of Information: Poststructuralism and Social Context", Polity, Cambridge.

Rorty, R., (1980), "Philosophy and the Mirror of Nature", Oxford University Press, Oxford.

Stowell, F., (1993), Hermeneutics and Organisational Inquiry, *Systemist*, Vol. 15(2), pp. 87-103.

Weber, M., (1993), The 'anything goes' philosopher. *Times Higher*, Vol. 1101, p. 15.

Winfield, P., (1991), "Organisations and Information Technology: Systems, Power and Job design", Blackwell, Oxford.

Winograd, T., and Flores, F., (1987), "Understanding Computers and Cognition", Addison-Wesley, Reading U.S.A.

Zuboff, S., (1988), "In the Age of the Smart Machine", Heinnemann, Oxford.

IMPACT OF SOLUTIONS IMPLEMENTED IN ORGANIZATION SYSTEM ON INFORMATION SYSTEM

Miro Jeraj

University of Maribor, School of Organizational Sciences,
64000 Kranj, Prešernova 11, Slovenia

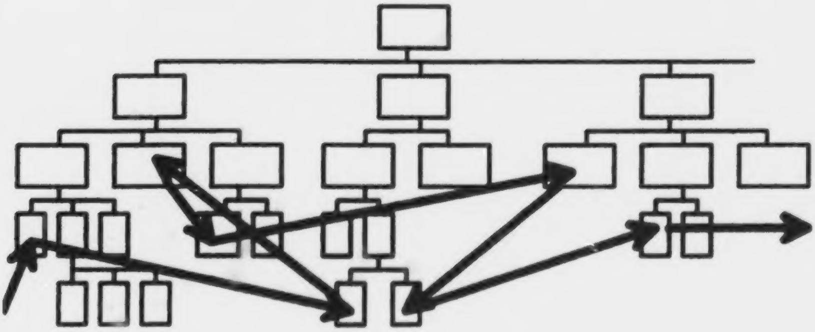
Abstract

The entire organization system of a concrete business system is split into a variety of processes joining into integral functioning of a company, the condition being, that all its processes really function well on the basis only of integrally defined information flow. This means that an efficient operation of a concrete business system is conditioned by a well-functioning organization system and an complete (linked, integral, complex) information system built for it

1. INTRODUCTION

Business system (Fulmer,1988) of a concrete company functions as a process (1), better said, the entire business process of the concrete company is split into variety of processes which join into the common functioning of an organization (Jeraj, 1991) Concerning the fact that individual process can be defined as a connection of interlinked tasks-operations, which have to be executed as to get final output; these tasks are allocated on working places of various jobs, departments and functions, the business process must operate without barriers (see fig.1). Processes (2) within organization system are used as support to business functions (3) of a business system.

As each task can start on the basis of some organization input which must come from somewhere, the process actually operates only on the basis of well defined integral information flow (Jeraj, 1992), which is or is not supported by appropriate information technology. Or inverted, if we want to build up a complete (linked, integral, complex) computer unsupported or supported information system of a concrete business system, we must first settle its organization system



Figures 1 Presentation of the concrete company process (J.Martin,1993)

2. GIVING PRIORITY OF BUILDING UP BUSINESS ORGANIZATION SYSTEM TO BUILDING UP BUSINESS INFORMATION SYSTEM

Building up business organization system and business information system of a concrete company is a complex issue which has to be tackled at the same time from the basic, managerial and informational aspect (Kajzer,Kavkler, 1987). Several analyses of successful companies in the world showed that the most successful companies are those which were able of a radical and multi-layered transformation (Sauerbrey,1989) and adapted as much as possible to the updated environment of the company. The referred to researches without exceptions prove, that the biggest quality step has been made by those companies, which were the first and most radical to innovate managerial process and via it its organization system of business system. In favour of this statement there is the fact, that leaders must anticipate the changes of the state in the environment and within business system and use these statements for development of business and its organization, as well as information system. Each delay in this field disables taking an advantage of the action , affecting thus directly the efficiency of business system.

This actually means that investments into the basic process (production) and information process (purchase of information technology) don't bring about the wanted results, due to the fact that without adequately developed management process, being integral part of organization system, we can use neither the disposable production and installed information technics and technology nor innovations in basic and information process.

Therefore the statement, that our companies are to great extent unsuccessful and inefficient, due to inappropriate information systems, is wrong, so we should not seek "a scapegoat" (Kajzer,1992) in merely badly suited information needs of managers,e.g. in inefficient information systems, but mainly in badly defined information needs. This, by all means, is a result of undeveloped management process and the effected organization system of a concrete business system ,as well as the mistakes in defined and performed processes.

The present managements of companies (not in our country only) are very often not aware enough of the necessity to define the organization and information strategy and its integration into business strategy of a company (Rant,Jeraj, Ljubič, 1992). If companies had defined objectives and targets concerning organization and information strategy and tactics, they would avoid many a failure in this field.

3. WHY ARE PROCESSES, RUNNING IN ORGANIZATION SYSTEM, TO BE SETTLED AND COORDINATED

3.1. Characteristics of a well-functioning organization system are:

1. The main characteristic of organization systems is that the objective is set to them. Due to this reason subsystems of a defined organization system are selected as to enable reaching the set objectives. The basic condition to reach each individual objective is system operation. Each subsystem here has its own function, connections among functions of individual sybsystems must be coordinated. This is achieved by a right and optimally defined process running in the business process (Fig.1). Herewith we should know that permanent operation of business system isn't possible unless organization system and its processes are managed. According to this fact, within organization systems, a new partial system of the organization system arises, treating this one from the management point of view.

2. We have to stress the need to build up business organization system before building up business information system and dynamics which reflects in the following:
   - permanent movement is typical for each organization system.
   - each element of organization system must be observed timely as well as spacely
   - dynamics must be understood as transforming quantity to quality because transition from quantity to quality means transition from one system state into another state
   - dynamics should be understood as movement and development. Separating of these two independent categories isn't acceptable because organization systems, mainly, are submitted to permanent new inputs into system and new outputs from system. The result of this movement is reflected in the development stage of the system itself.

   All the quoted features of dynamics are mainly reflected as a direction and intensity of movement, as well as the changes in the structure of the business system itself.

3. Dynamics of organization systems is shown, above all, in the influence of environment on specific system elements and the influence of system elements on environment. Hence it follows, that between system elements and environment certain relations arise. These relations among system elements (and their

environment) are important for balance and stability, as well as efficiency and good performance of the business system This is especially valid for business systems which, due to environment changes, change their strategic goals of business making activity These changes to a great extent affect the dynamics of changes in the organization and information system

3.2. How to realize the set requirements arising from the described characteristics for a well-functioning organization system

Management of the business system daily faces the task described above They have to decide whether to search their chances in the business system or in the environment When deciding for searching their advantages in the business system, they will be searching them within the frame of investments, operation, products, information or management process Should they search for advantages within the environment of business system, this will be then searching for the relationship of business system with the competition items and suppliers, buyers, business systems producing substitutes, with beginners or competitors, respectively The management must, therefore, define the so called strategic goals where they will focus on strategic jerks supported by the use of information technology which would assure competitive advantage to the business system

In order to assure all what is necessary to achieve the set objectives, the management of the business system must eliminate the chaos in the organization system arising from the following

- managers don't have the task to change the business functions or departments, they have to change the work in processes which has to be implemented by the staff as to achieve the set objectives This means that management has to settle processes in its organization system because this is the only way to perform tasks,which have to be implemented, successfully

- the anticipated work is not implemented by business functions or departments, this work, however, is implemented by the employees responsible for a concrete task in the concrete process Business functions and departments are visible,actual, known by all the employees, contrary to the process being run in organization system, management doesn't know much because they are invisible, hidden behind organization structure, they don't have their own names and they don't have their own leaders to manage them

- the managers who we know, manage business functions or departments, none of them is given responsibility to perform the whole work,e g the whole process The success and efficiency of the business system depend on the result (achieving the set objectives) of the concrete process

- to enable well-functioning of a concrete organization system and the processes run within it, we have to be acquainted with them in details This means that we have to define clearly where the start is and where an end of a certain process is

- to be able to speak about a well - functioning system, we have to know the advantages of process schemes (f.ex. process schemes made on the basis of a block diagram, etc.) which show us the flow of work through the company (see fig.1). By means of this, we will help employees to discuss among them the vital changes of a concrete process and at the same time a chance will be given to the management to achieve the set objectives.

3.3 Planning changes of organization system and the applied information technologies

If we want to apply information technology to achieve competitive advantage of the business system, we must include it into the process of business planning (Rant, Jeraj, Ljubič, 1992). This means that planning of business, organization and information systems has to be joined (Synott,1987, Lucas, 1988) into integral plan. Lucas herewith states that there were few business systems which entirely succeeded in it. According to our experiences, it is not the case that there wouldn't be any planning in the business systems, the problem is that there isn't enough knowledge (management must set the requirements and define content) in the business systems to apply results and experiences for the needs of such planning. This knowledge, namely, arises from a detailed knowing and anticipated operating of its own organization system and the processes running within it. If processes are badly defined, business tasks in them can't be defined either; often, on such a basis, we can neither define global nor information needs for this concrete company.

4. CONCLUSION

Business system can, on the basis of well-functioning organization system and the processes running within it, define its information needs. The information needs, thus defined, are the basis of carefully planned information system development in the company, special stress being given to a uniform and integral database. Uniform and integrally built database, namely, enables connection of managerial and operational information system. It is worth mentioning that it is only this way that business system can avoid chaos, necessarily arising from the use of new information technology, when being performed in the old uncontrolled way.

5. REFERENCES

Fulmer,R. The New Management, McMillan Publishing Company, New York, 1988, page 132.

"A company ... is a process of people working together for achieving a common objective. This definition contains four consisting concepts:

1. that a group of people is needed what nobody can achieve alone (sometimes mentioned as synergy -M.J.)
2. that they must work together to combine their talents and skills
3. that sharing work groups people into their interests and skills
4. that uniformity of targets directs their individual efforts

James Martin: Documentation for the World Seminar, Published In U.K. by Savant Institute, 2 New Street, Carnforth, Lancashire, 28th Edition, 9th Series,1993. B.9.15.

Jeraj,M.: Definiranje modela povezanega enotnega računalniško podprtega informacijskega sistema za proizvodne poslovne sisteme s posebnim poudarkom na povezanosti tehnično proizvodnega področja z drugimi, doktorska disertacija, Univerza v Mariboru, Fakulteta za organizacijske vede Kranj, Kranj, 1991

Jeraj,M.: Organizacijski vidik prenove informacijskega sistema, V: I. Posvetovanje doktorjev in magistrov organizacijskih ved, Brdo pri Kranju, 10. in 11. decembra 1992, Fakulteta za organizacijske vede, Kranj, 1992, str.78-82, in Jeraj, M., Rant, M.: Prenova podjetja - organizacijsko informacijski vidik, V: 1. Mednarodno posvetovanje o prenovi informacijskih sistemov - RE 92, SRC Ljubljana, Bled, 1992, str. 77-80

Kajzer, Š., Kavkler, I.: Kibernetika ekonomskih sistemov I., Univerza v Mariboru, VEKŠ, Maribor, 1987

Kajzer, Š.: Izhodišča za oblikovanje sodobne zamisli o podjetniškem raziskovanju in prognoziranju, XIX. Posvetovanje o podjetniškem planiranju, Portorož, 21 do23. oktobra 1992, V: IB reviji, st./8-9-10/, letnik XXVI/1992, str.5-13.

Lucas, H. C. Jr.: Integrating Information Technology and Strategic Planning:Coping With Two Paradoxes, Center for research on Information Systems, Information Systems Area, Graduate School of Business Administration, New York University, Working Paper Series, CRIS #174 GBA#88-21, March, 1988, str.2.

Rant, M., Jeraj, M., Ljubić, T.: Enoten kompleksen organizacijski sistem planiranja v proizvodnih organizacijah, POIS Radovljica, 1992, str.45-52.

Sauerbrey,G.: Betriebliche Organisation in Informationszeitalter, dr. A. Huethig Verlag Heidelberg, 1989.

Synnot, W. R.: The Information Weapon:Winning Customers and Markets With Technology, John Wiley&Sons Inc., New York,1987, str. 44-46.

(1) A process is dynamic sequence of phenomena in nature, society or thinking which follows in time;majority of processes is irreversible - they can run in one direction only; Pojmovnik poslovne informatike, 1987, str.228.

(2) Processes running in organization system could be called organization processes. In the process structure of business system we face a business process, organization processes, partial organization processes and elementary organization processes (tasks, which, when executor is known, are connected into organization tasks).

(3) Business function is integrity of identical or similar, mutually linked operation, which means clearly defined fields of work within a business system, with defined parts of all tasks in it. Pojmovnik poslovne informatike, 1987, str.197.

Strategy Planning

MAIN FUNCTIONS OF STRATEGIC INFORMATION SYSTEM

Sonja Treven

University of Maribor, School of Business and Economics Maribor, 62000 Maribor, Razlagova 14, Slovenica

Abstract:

In this paper the author emanates from the strategic management and the advantages of its implementation in an enterprise. Furthermore, she demonstrates the basic features of the strategic information system that ought to provide top managers with prompt information for the needs of strategic management, and deals with the functions that must be carried out by any such system to achieve the purpose of its development and introduction into an enterprise.

1. INTRODUCTION

Nowadays computer terminals and personal computers cannot be considered an exception in an enterprise. Officials and secretaries have used them for years. Operating managers as well as the middle management in the enterprises are also discovering their apllication value and therefore this new means of information technology is being used more and more in their planning, decision making and control activities.

The idea of top management ever spending their time behind terminals used to conjure a smile on their faces even some years ago, or caused shaking of their hands. The image of executives "working hard" behind terminals was not they would cerish at all.

Managers at the highest organisational level make decisions expecially about the problems that are of strategic significance for their enterprise. They also carry out the activities connected with the process of strategic management. For such activity they need information from the environment (competition, science and technology, political and economic system) as well as the information about the factors regarding the enterprise (finance, marketing, staff, culture, etc.). Becouse of their importance the information can be treated as "strategic" information and the system which assures such information to users - top managers can be characterised as strategic information system (SIS).

The basic aim of the strategic information system is, with the computer support, to ensure more effective use of the information in the decision processes appearing with strategic management of the enterprise.

The term SIS can be used for "systems supporting strategic management" as well as for "systems that support or shape the competitive strategy of the firm". In this article the term SIS is used according to the first definition. An alternative term we could use is "Executive Information Systems".

In the paper we will emanate from strategic management and the advantages of its implementation in an enetrprise. Furthermore, we will demonstrate the basic features of the SIS that ought to provide top managers with prompt information for the needs of strategic management. Finally, we will deal with the functions that must be carried out by any such system to achieve the purpose of its development and introduction into an enetrprise.

2. THE NATURE OF STRATEGIC MANAGEMENT

The concept of strategic management appeared in a literature in the field of business management in the middle of the seventies. The uniform definition of that concept has not existed until today. The greatest contribution in developing that idea and it's affirmation is ascribed to H.I.Ansoff who defines strategic management as a systematic approach to a major and increasingly important responsibility of general management: to position and relate the firm to its environment in a way which will assure its continued success and make it secure from surprises (Ansoff, McDonnell, 1990).

Wright, Pringle and Kroll interpret the strategic management as a series of steps in which top management should accomplish the following tasks (Wright, Pringle, Kroll, 1992):

1. Analyze the opportunities and threats or constrains that exist in the external environment.
2. Analyze the organization's internal strengths and weaknesses.
3. Establish the organization's mission and develop its goals.
4. Formulate strategies (i.e., corporate-level, business unit-level, and functional-level) that will match the organization's strengths and weaknesses with the environment's opportunities and threats.
5. Implement the strategies.
6. Engage in strategic control activities to ensure that the organization's goals are attained.

3. BENEFITS OF STRATEGIC MANAGEMENT

An enterprise can attain several benefits from appropriately implementing strategic management. Perhaps the most important benefit is connected with the tendency to increase its profit (Certo, Peter, 1990). The researches in recent years have also confirmed the assumption that an effective and efficient strategic management system can rather increase the profitability of an enterprise.

Besides financial an enterprise can gain other distinctive advantages stemming from the strategic management system implementing. Some of these are the following:

1. Warns the problems may appear before they happen.
2. Prepares the enterprise to changes and allows its action in reaction to changes.
3. Assists managers to increase their interest in an enterprise.
4. Enables managers better understanding of business.
5. Allows an objective view of management problems.

6. Provides a basis for valuating the execution of the plan and for controlling activities.
7. Reduces the negative effects of inconvenient circumstances and changes.
8. Suggests more effective allocation of time and resources as to knowed opportunities.
9. Creates a framework for internal connections among personnel.

Of course, these benefits do not appear in any enterprise that posesses a strategic management system. They are accrued only if such a system is implemented in an effective and efficient manner in an enterprise (Certo, Peter, 1990).

4. FEATURES OF STRATEGIC INFORMATIONS SYSTEMS

Information provided by strategic information system to top managers and pointed out as "strategic" information from the beginning of our text have certain joint characteristics. Let us present some of them:

1. Strategic information usually describe events, appearances, systems, people and are, therefore, to a large extent qualitative, only some of them are quantitative.
2. Strategic information are highly subjective as the personality of people, gathering and formating them, have a great impact.
3. Strategic information may accrue in verbal, written or optical form and are, therefore, reported with speach to users in a case of their indirect contact with transmitter of information; in a form of printed texts or graphical presentations.
4. Strategic information refer, by their content, at most to happening in the environment and less to what is taking place inside of the enterprise.
5. Strategic information is a condensed information with the low grade of details.

However, the essence of strategic information systems is not only in providing information to the highest level of managers, but in supporting their day to day functions, too. Hence, we have to treat these systems in broader sense as they enable: (Bobek, 1992):

(1) searching of data about the enterprise (aims of enterprise, accomplishment of planned results, etc.)
(2) searching of data about the relevant environment (competitors, suppliers, consumers, etc.)
(3) solving of decision problems
(4) presentation of decisions to co-workers and business partners
(5) simple and effective communication with other managers and co-workers.

First strategic information systems were oriented to the individual application and were created for individual managers. Today in enterprises of high developed western countries even operate the systems developed for the support of whole management.

For example, in Philips Petroleum Company all nine top managers have access to daily information about the transormation of oil and derivate sale as well as the summaries of international news they completed them three times a day. In Xerox Corporation all managers are connected via computer network. Each of them has also access to data needed for managing his(her) field of work.

5. FUNCTIONS OF STRATEGIC INFORMATION SYSTEM

Top managers need information from the environment and from the inwardness of the enterprise for creation and choice of strategy as well as for the evaluating of implemented strategy. The strategic information systems have to gather these information in their "raw" form, in the form of data, and then to analyse them after their significance for the users.

Data eliminated as important ones from the crowd of all data are then storaged in a database from which they are at disposal to users - top managers. With the aim users could get needed information on quick and simple manner strategic information system creates "data cubes" providing presentation of information in more than two tradicional dimentions. Besides the above mentioned way of assuring information, that system has also to enable top managers to gather information that are result of different analysis (i.e., finance analysis, competition analysis, market analysis, etc.).

All above mentioned activities can be summarized into four functions every strategic information system has to execute to reach the aim of it's development as well as it's implementation in an enterprise. These functions are:

1. Gathering and analysing of data
2. Storage of data and models
3. Creating of "data cubes"
4. Reporting top managers.

5.1 Gathering and analysing of data

As we already know different factors from the environment as well as from the inside of an enterprise have the impact on strategic decisions. For this reason the strategic information system has to assure the executing of activity of acquiring the data related to both groups of factors.

The data collected from the inside of the enterprise are numerically limited since they arise from the externel state of the enterprise while the amount of data collected from the environment is bigger. The environment an enterprise operates in is namely extensive and very diverse. The amount of data resources, the amount of data and the diversity of fields the data are related with is, therefore, much bigger in the environment of an enterprise than in it's inwardness. But the natural need of an enterprise is to dispose with data from all fields influencing on its existence and success.

Thus, a whole spate of data on the environment as well as on the enterprise ought to be collected whereby different modes of gathering could be used as well as various resources of the data (Turbain, 1993). The question put up immediately to us concerns how to organise that function.

In General Electric more than a hundred of people are occupied with the function of data gathering with the help of a great computer network. On the contrary, in small enterprises according to the size of the enterprise less people are employed dealing with this function. Thus, it is needed in relation to the size of an enterprise and the complexity of it's businesses to find a mode how we shall carry out this work as well as how many people will do it.

We have to utilize computers, public computer networks and public data bases (business bases, bases about patents, bases related to the special technical scopes, etc.)

with the execution of data gathering activity. There can also be used publications of different institutions and governmental organisations, industry spying, employment of people employed before with the competitors or enterprises influencing to the business of our enterprise and the data acquired by different economic associations.

Data gathered should be translated in the enterprise if they are, for example, in other languages and then the data must be the subject of analysis. The purpose of such analysis is to acquire the data needed by top managers from the crowd of available data and also to refine and condense the data.

The analysis of data is closely connected with the process of their gathering. Therefore, the link-up among the people involving in the process of data gathering have to be provided. In that process the cooperation of users (top managers) is necessary as they are frequently the best estimators of freshly collected data.

5.2 Storage of data and models

Data found as important for top managers and as that also eliminated from the crowd of data gathering should be then stored in the database. In order to make these data accessable to different top managers from the enterprise it is necessary to provide appropriate hardware and software.

The hardware has to comprise the external computer storage on which there is the database, the facilities, control units and channels allowing the connection of user with the data. Besides hardware, software is needed, too. It consists of the computer programmes which bridge the gap between the users requirements and the manner in which the data are organized and accessed on the computer storages. This software is called the database management system (DBMS) and permits the data to be stored in one place while making them available to different applications (Landon, 1993). Specifically, a DBMS enables entering (or adding) information into a database, updating, deleting, manipulating, storing and retrieving the information.

DBMSS are designed to handle large amounts of information. Often, data from the database are extracted and put in a statistical, mathematical, or financial model for further manipulation or analysis.

Similar as the data, different routine and special statistical, financial, management science and other quantitative models that provide the analysis capabilities in the strategic information system, should be storage in model base. Such models can be qualified as strategic models since they are used to support top management's strategic management responsibilities.

The application of models storaged in the model base enable the strategic information system users to:

1. Formulate different forecasts.
2. Find the optimal resources allocation of the enterprise.
3. Simulate time dependent factors.
4. Prepare rather reliable description of the future.

The application of models is enabled by data storaged in the database and by estimations of users. Herein the appropriate software which have to allow the access to models storaged in model base in the case of their usage, is necessary, too.

5.3 Creating of "data cubes"

Although, the strategic information systems distinguish one from the other among enterprises, each of them enables the creation of "data cubes". These data cubes can also be created differently depending mainly on the needs of users - top managers in enterprises.

Particular cube can involve, for instance, three dimensions of data (products, consumers, finance results) for only one enterprise, and for one time period. With greater number of cubes we can gain another dimension of data - time (each cube for one time period) presented in vertical coloumns of cubes in Figure 1. Fifth dimension referring to comparisons (raws of cubes) can be added for instant with cubes of scenarious for presentation of comparisons between competitive enterprises.

The formation of cubes with data enables top managers modelling and presenting of data in more dimensions (i.e., finance, markets, products, regions and time). When they use these cubes depends mainly on business situation in which they need more than two traditional dimensions.

Strategic information system enables managers at the highest organizational level quick retrieving of reports on the screen of their terminals including results of financial and market analysis, analysis on competition and products as well as another analysis. All of that is possible with the formation of cubes mentioned, and with the usage of various models. Besides, that system allows also access to more detailed data that are inside of the particular cube dimension.

5.4 Reporting of top managers

Strategic information system provides messages to top managers on two different modes:

1. with the access to data and reports about the current state of the enterprise or forseen trends of business
2. with the possibility of execution independent analysis on the ground of available data.

According to the first mode top managers acquire current messages about the state of important variables (i.e., sale, consumers, products, competitors, etc.) influencing on the success of their enterprise or about the expected trend of their movement. Furthermore, they can also acquaint with the critical business fields treated as "weaknesses" (i.e., the quality of products, costs of control, services for consumers, etc.).

Acquireing of messages according to the first mode is passing so that the top managers, comfortabley placed in the armchairs behind the computer terminals in their offices or at home, determine the type and the form of needed message. The requested message is then naturally presented to them on the terminal screen. In this way, top managers gain the messages quickly, simply as well as with small costs and risks.

However, strategic information system does not enable top managers the usage of computer to acquire the messages only according to the first mode. It gives them also the possibility for the execution various analyses on the ground of data storaged in the database.

According to the second mode, top managers can acquire the messages that are the result of various analyses, as for example the analysis of strenghts and weaknesses of enterprise, the analysis of opportunities and threats, etc. Besides these, top managers can execute another kinds of abalyses too, for example, the analysis with the help of simulation models (i.e., the analysis of finding the scopes, the investment of capital), the analysis on the ground of usage the forecasts models (i.e., the extrapolation of current trends into the future) or they can create certain trends in grafical form and gain thatway the additional visual sight on the topic they are interested in.

As the highest level of managers could acquire mentioned messages they have to know at least a little the scope of computer programming. They can use query languages allowing the execution of computer programms for gaining messages from the database on the ground of their requests. These languages can be used without the knowledge of programming techniques. With the help of such languages top managers have the access to the needed messages also without intervention of professional staff.

6. CONCLUDING REMARKS

Strategic information systems naturally diverse among enterprises. If these systems achieve the aim becouse of it they are introduced in particular enterprise, it can be judged on the ground of top managers' following statements:

(1) "SIS spare planty of time I will use it otherwise for communicating with the staff of individual functional scope. Today, I can order for more and more problems together the needed data and the form of report I wish to receive the required data in. They are then sent to me in much shorter time as my description of information needs to the appropriate specialist would required."

(2) "SIS enables me to make on the ground of various comparisons between our enterprise and the competitive one a quick definition of the scopes that need the most of my attention.

(3) "Some of my best ideas appear between 5 p.m. and 7 a.m. Therefore, the access to the relevant data in that time for varifying certain facts, is very important to me. The terminal connected from my home with the computer in the enterprise enable me to execute the analyses yet in time the idea is still "fresh" and in front of my attention."

(4) "My way of thinking is graphical. SIS allows me to acquire the data in the form most suitable to me, that is in graphical form."

7. REFERENCES

Ansoff I., McDonnel E., "Implementing Strategic Management", Prentice Hall, 1990, pp. XV.

Bobek S., "Organiziranje sistemov za podporo direktorjev", Organizacija in kadri, Vol.1-2, 1992.

Certo S.C., Peter J.P., "Strategic Management: A Focus on Processes", McGraw-Hill, 1990, pp. 7.

McGee J., Prusak L., "Managing Information Strategically", John Wiley & Sons, Inc., 1993.

Landon K.C., Landon J.P., "Business Information Systems", The Dryden Press, 1993, pp. 194.

Treven S., "Delovanje strateškega informacijskega sistema", Organizacija in kadri, Vol.1-2, 1993.

Turbain E., "Decision Support and Expert Systems", MacMillan Publishing Company, 1993, pp. 132-134.

Wright P., Pringle C.D., Kroll M.J., "Strategic Management", Allyn and Bacon, 1992, pp. 4.

A PROCESS MODEL FOR INFORMATION SYSTEMS STRATEGY PLANNING

Gregory Mentzas

Assistant Professor, Department of Electrical and Computer Engineering, National Technical University of Athens, 42, 28th October str., 10682 Athens, Greece

Abstract.

Although information systems strategy planning is vital to continuing organisational sucess and despite the existence of a multitude of frameworks and methods, organisations are still failing to deal effectively with IS planning problems. In order to help alleviate this problem, the present paper proposes an Information Systems Strategy Planning (ISSP) process. The paper identifies separate process elements: phases, stages and modules of elementary activities. The proposed process aims to satisfy the need for a consistent linkage of IS strategy with corporate strategy by adopting elements of the corporate strategic planning literature. It provides a framework for enforcing the participation and involvement from managerial staff of the target organization and supports a structured team approach during the process's implementation. Finally, it includes elements from the quality management approach, in order to guarantee that review and control activities are explicitly embedded.

1. INTRODUCTION

Information systems strategy planning (ISSP) has been identified as vital to continuing organisational sucess and effective information systems performance; see e.g. Hartog and Herbert (1986), Brancheu and Wetherbe (1987). However, although ISSP methodologies have been used for many years, organisations are still failing to deal effectively with untangling IS planning-related problems. This is mainly because many ISSP methodologies do not seem to realise that the IS-related problems result not merely from technological problems but are also derived from paying too little attention to the interrelationship between IS and business and organisational factors; see Galliers (1991).

Research has shown that despite the growing number of theoretical frameworks for ISSP, practice still faces three major problems:

- need for integration of ISSP and overall corporate strategy; see e.g. Lederer and Sethi (1988), Galliers (1991) among others;
- moderate practical utility of existing ISSP methodologies; Lederer and Sethi (1988);
- limited management involvement and committment to the ISSP activities; Galliers (1991) and Lederer and Sethi (1992).

It has been argued by Lee and Gough (1993) that the multi-dimensional nature of the ISSP process must be considered. This focus on processes is in line with recent research work in software process modelling, see e.g. Curtis et al (1992), as well as with the effort on total quality management; Deming (1982). Such efforts highlight also the need for structuring the work of teams.

This paper attempts to develop an information systems strategy planning process (ISSP) model, that aims to increase the practical usefulness of information systems planning within organizations. The process model identifies separate elements of the process: phases, stages and modules of elementary activities, and aims to satisfy the need for a consistent linkage of IS strategy with corporate strategy by adopting elements of the corporate strategic planning literature. It also provides a framework for enforcing the participation and involvement from managerial staff of the target organization and supports a structured team approach during the process's implementation. Finally, it includes elements from the quality management approach, in order to guarantee that review and control activities are explicitly embedded.

The paper starts with a review of the literature on information systems strategy planning. The various frameworks and methods proposed are presented and the focus on the planning process is highlighted. Then some important topics in the planning process are examined; they include the linkage of business and IS strategy, the emphasis on the participation of management, the support of team work and the inclusion of review and control activities. Next, the phased approach to the ISSP process is presented, together with the organizational structure and the related management procedures. Finally, the concluding remarks identify the advantages and weaknesses of the approach and point to directions for further research.

2. REVIEW OF THE LITERATURE

2.1 Information Systems Strategy Planning

Information systems planning (ISSP) is the first stage in the system development life-cycle. Olle et al (1991) indicate that ISSP is taken to refer to the strategic planning of computerised information systems. The main purpose of ISSP is to identify which information systems are needed rather that planning in detail for any specific system. An information systems plan should show which new systems are required and the sequence in which they should be implemented.

Several terms have been used to represent the strategic level of information systems planning; e.g. Strategic Information Planning (SIP), Information Strategic Planning (ISP), etc. Although these terms include the strategic aspect of information systems planning, they all seem to view the process as context-independent. Exceptions to that include Galliers (1991) and Ward et al (1990), who view ISSP as a management task concerned with:

- integrating information systems considerations into the corporate planning process,
- planning for effective long-term management and optimal impact of information, information systems and information technology;
- incorporating all forms of manual systems, computers and communications; and
- addressing the problems of limited management awareness, communication barriers and problematic organisational approaches.

The challenge of ISSP is to develop an integrative process that helps managers identify the potential of IS, matches the right IS projects to the company's business plans, facilitates communication between IS, users and management and ensures that sufficient resources are available and used optimally. Many frameworks and methods have been proposed towards this end.

2.2 Frameworks and Methods for ISSP

In order to implement an ISSP study, an organization usually selects an existing methodology and embarks on the study, or uses external ISSP consultancy, in which case it adopts the ISSP vendor's methodology. A significant number of approaches have been developed in order to assist the ISSP process. Surveys have been given in Kim and Michelman (1990), Munro and Huff (1985) and Neo (1988).

2.3 The ISSP process

A recent change in the development of ISSP methodologies has occured, towards multiple methods. For example, Galliers (1991) in reviewing the development path of ISSP methods stresses the need for '*multiple methods*', i.e. the fact that elements of different methods are likely to be more or less required in different cisrcumstances and that ISSP should include human, organisational and infrastructural issues. Other researchers have also found that there is no one single way for undertaking information systems strategy planning and that a combination of approaches and tools may be required; Earl (1989), Ward et al (1990). In addition, a study based on a survey of companies in the UK suggests that the focus of ISSP should no longer be on method alone; rather, the multi-dimensional nature of the ISSP process should be explicitly considered; see Earl (1990).

Premkumar and King (1991) defined the IS planning process as comprising of "context", "input", "process", "output" and "outcome"; see Figure 1. Given the importance of the participation of executives from the target organisation in the planning process, some educational and training activities are also considered necessary, to help the participants understand the planning process and the potential of IS technology in their operations. The resources required refer mainly to human, computer, time and cost resources to be used during the strategy formulation exercise. The importance of e.g. finding a respected business veteran, comfortable with current technology as the team leader, or of appropriately scheduling interview times, especially for the highest paid managers, have been considered crucial issues in the sucess of the planning exercise; see Lederer and Sethi (1992).

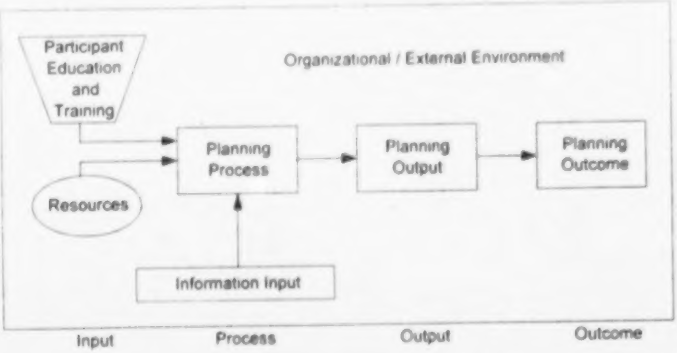


Figure 1. Information Systems Planning Model (amended from Premkumar and King, 1991)

Although a "multiple methodology" approach has been widely canvassed for the development of ISSP methodologies, all current ISSP methodologies seem somewhat normative, in that they do not adress managerial problems that arise as to how strategic planning is to be done and how to make the best use of current methodologies; in other words, they focus on the issue of "what should be done", rather than on "how to do it".

3. TOPICS IN INFORMATION SYSTEMS STRATEGY PLANNING

3.1 Integration with Business Strategy

The need to integrate business and IS strategy has been stressed by researchers for more than two decades; see e.g. Kriebel (1968). Information technology has been analysed as a strategic tool for enabling competitive advantage, as well as a device for sustaining this advantage. Most of the ISSP methodologies make explicit reference to the overall business strategy and its relation to IS strategy. In addition, a number of studies attempt either to integrate IS and business strategy or to provide frameworks for identifying opportunities for incorporating IS strategy within an overall corporate competitive strategy. Finally, other studies aim to provide recommendations for grounding IS strategy to findings from corporate strategy research.

Nevertheless, several surveys of IS managers' opinions and studies of the sucess factors in ISSP show that the alignment of IS and corporate strategy is still one of the top priorities for the future; see e.g. Lederer and Sethi (1988) and Galliers (1991). In an attempt to integrate and align IS with business strategy the process described in this paper harmonises the formulation of IS strategy within five generic corporate strategy phases; see also Thompson (1993). In addition, in order to overcome the problem of involvement of managers it provides for their explicit participation.

3.2 Participation Aspects

The lack of management involvement, as well as committment, to the ISSP activities has been considered as one of the major problems for the formulation and acceptance of an IS strategy; see e.g. Galliers (1991) and Lederer and Sethi (1992). One way of solving this problem is to classify the various possible roles of management involvement and examine how these roles can be included in the strategy formulation process.

In a discussion of alternative ISSP processes Ormerod (1992) distinguishes between two styles: the "Conventional Process", in which the members of the working group carry out interviews of the organization's managers; and the "Participative Process", in which members of the organisation's management join the working group in the actual technical work. Within the participative process five prototypical roles, first identified by Friend and Hickling (1987) in the "strategic choice approach", are used. These roles are: "accountable" for the decisions to be taken; "responsible" for guiding the conduct of the decision-making process; those to whom periodic "reference" should be made because they have crucial roles in other fields of decision-making; those who may have a "representative" in relation to specific interests; and the "stakeholders" in the sense that they will be directly impacted by the decisions taken in the strategy formulation process.

The process described in the next section aims to satisfy the participation objective, based on the premise that ISSP processes that ensure the involvement and committment of the target organisation's executives are much more effective, although they can be more demanding of management time. However, a less complicated distinction of user roles is made, compared to the one by Friend and Hickling (1987). The organisational structure and the participatory scheme is presented in section 4.2.

3.3 Team-work Aspects

It has been argued that the strategy formulation process is an inherently group process, requiring a shared understanding of a wide range of world views and assumptions on the part of key stakeholders, both within and outside the organisation concerned; Galliers et al (1991).

Recent attempts in Group Decision Support Systems (GDSS) and Computer-Supported Cooperative Work (CSCW) aim to aid group decision-making processes in a multifold of approaches; see e.g. DeSanctis and Gallupe (1987) and Kraemer and King (1988). Such research efforts have been recently extented to the area of strategy formulation. The goal here is to electronically support the various tasks during strategic planning efforts effected by groups. An important issue within this framework is to guarantee the coordination of joint work; see Malone and Crowston (1990) and Mentzas (1993).

Another research direction can be found in the team approaches related to software development; such work aims to facilitate, organize and manage the software development process; Rettig (1990). The process described in the next section is based on the assumption that the key stakeholders, as well as external participants, collaborate during the strategy formulation process. The approach used is that of small teams, in a vein similar to that proposed by Rettig and Simons (1993) for software development and Galliers et al (1991) for corporate strategy formulation.

3.4 Review and Control Activities

Many discussions of strategic planning do not extend beyond identifying issues, stakeholders and objectives; see e.g. Porter (1980). These discussions usually focus on the perspectives of Chief Executive Officers and senior staff of an organisation. However, emphasis should also be directed to the review activities of the strategic plan formulation process. Such activities are considered necessary to assure achievement of the plan.

In addition, it is of importance that a strategic plan formulation process should include planning for control activities. These control elements would evolve from a set of objectives which specify metrics and review dates. During the evolution of the plan, development changes may occur and may dramatically alter the relationships between objectives, control metrics and stakeholders. Hence the process should provide a flexible structure that supports these elements.

A common control cycle described in Total Quality Management (TQM) literature is the "Deming Circle" or "Plan-Do-Check-Act" (PDCA) method; see Deming (1982). The last two steps are absent from many discussions of the strategic planning methodology. The Check step represents the comparison of the results of the process to the expected goals. Any deviation is then analyzed in the Act step, where causes are assigned and corrective actions are identified. A similar idea is captured in an oft-quoted phrase by Ackoff (1981): "A good deal of the corporate planning I have observed is like a ritual rain dance; it has no effect on the weather that follows, but those who engage in it think it does". The metrics of a planning process must be monitored and the relationships between coordinated activities compared, to assure that the planning process is progressing as desired.

It is obvious that the need for review and control activities is fundamental also in the case of information systems strategy planning. In the proposed ISSP process we adopt a revised version of PDCA that takes into account the team dimension of the planning effort and places emphasis on review, revision and approval procedures.

4. AN ISSP PROCESS MODEL

4.1 Basic Framework of the Process Model

A process can be defined as a set of partially ordered steps intended to reach a goal; see Curtis et al (1992) p. 76. Any component of a process is considered a process element. A process model is an abstract description of a process that represents selected process elements that are considered important and can be enacted by a human or a machine. Planning the IS strategy formulation process beforehand can prove to be a tedius and even dangerous task. Hence a "divide and conquer" approach is proposed, as in Rettig and Simons (1993).

The decomposition distinguishes between three process elements; they are the following, by order of increasing detail: phases; stages and modules. Such a decomposition can guarantee the modularity of work, assist in the proper work allocation and facilitate the overall management of the process. In order to accomodate the major target of the ISSP process, i.e. the consistent integration of business and IS strategy formulation, the phases of ISSP are generic strategy formulation steps that can be applied to any corporate strategy development process. Each phase is divided into stages. Stages are considered to be semi-autonomous components of work, which can be planned relatively independently. The stage is defined in terms of the resulting behaviour and appearance of its end-product and the information structures that underlie it. Stages are further divided into modules. Modules can either be units of work (i.e. activities) or collection of activities.

Each of the above process elements (phases, stages and modules) has its own objectives (i.e. purpose within the overall process), participants (i.e. the recommended actors and their roles), preconditions (i.e. inputs, references, etc), products (i.e. the end products subject to management and quality control) and techniques (i.e. the models and tools that may be used).

Finally, following the related literature on the application of Total Quality Management methods in information systems development, each module of the ISSP process goes through a mini life-cycle with five milestones: Plan; Approve; Do; Review; Revise; and Evaluate.

The following sections expand on the main elements of the ISSP process.

4.2 Description of Phases, Stages and Modules

The first division of the overall ISSP process is into five phases (see also Table 1): Strategic Awareness; Situation Analysis; Strategy Conception; Strategy Formulation; and Strategy Implementation Planning. These phases are then separated into stages and modules of activities.

Table 1. Phases of the information systems strategy process

Phases	Examples of major issues
Strategic Awareness	Where are we going? How are we doing? Analysis of competition
Situation Analysis	Definition of opportunities and threats Definition of strengths and weaknesses Use of information as a strategic resource Analysis of information technology use
Strategy Conception	Where do we want to go? What strategic alternatives are available? Which are the elements of a good choice?
Strategy Formulation	Analysis of alternative growth strategies Analysis of business systems in alternative growth strategies Analysis of information systems in alternative growth strategies
Strategy Implementation Planning	Structure of the information systems strategy Resource management and control Risk management Management of change

The objective of the first phase is to raise awareness on the issue of strategy formulation and to provide first, rough answers to major questions concerning the overall direction of the organization and its competition. Hence the identification of corporate strategic goals is of crucial importance, since they will provide the quidelines for the development of and the alignment to IS strategy goals. In this phase the major collections of business processes and IT systems should be identified and the strategic relevance of each of these should be analysed. In addition, this very first phase provides for the overall planning of the process and the management and organisation of the process's implementation.

The phase of situation analysis can be classified in various stages concerned with the business, the organisational structures and the information technology issues. The latter issues should be examined both internally within the target organization and externally, i.e. with regard to the trends in the environment (e.g. competition, state of the art, etc). The major aim of this phase is to arrive at a clear and documented diagnosis of the existing business and IT situation in the target organization, to identify problems, misfunctionalities and inefficiencies, and in the meantime identify possible opportunities from the internal and external environment. Tools and techniques such as e.g. SWOT analysis, portfolio analysis, stage of growth models, analysis of competitive forces and value chain analysis are very useful. The aim is to synthesize a range of views as to the strengths and weaknesses of the organization and its aims and objectives, in the context of the environmental impositions and trends and its historical setting, all in the context of its management and planning of the information resource; Galliers (1993).

Strategy conception, the third phase of the process, can be considered including (at least partly and informally) the strategy thinking, analytical component of strategic decision-making;

see Mintzberg (1993). Strategy conception refers to scanning the future for the identification of opportunities for competitive and performance advantages and the identification of alternative scenarios for future growth. Alternative perspectives and assumptions should be identified and incorporated into the scenarios. In addition, a shared understanding and vision should be the overall objective rather, than a consensus. In this phase the IS scenarios to be constructed should be in accordance with the overall business strategy and should provide clear information technology insight to the efforts for their accomplishment. Techniques that are of use here include analysis of critical success factors, what-if analyses, cross impact analysis, scenario evaluation methods, multicriteria techniques, etc.

The other two phases (Strategic Formulation and Strategy Implementation Planning) conform with what Mintzberg calls strategic programming; actually strategy formulation, i.e. the fourth phase of the ISSP process, corresponds to what Mintzberg (1993) calls codification, i.e. the clarification and expression of strategies in terms sufficiently clear to render them formally operational., while implementation planning corresponds to elaboration and conversion, where elaboration is defined as the breaking down of codified strategies into substrategies and ad hoc programs as well as action plans, and conversion is the consideration of the effects of the changes on the organization's operations.

In the strategy formulation phase the chosen scenario(s) should be analysed in terms of the functions (business systems), hierarchies and responsibilities (organisational structure), as well as in terms of the technical architecture required for the building of IT systems that would support the alternative growth strategies. Hence, in this phase various models of the target organization should be elaborated: a functional model depicting the flow of information; an organizational model depicting the responsibilities and hierarchies; and a technical (logical and physical) model depicting information storage, distribution and processing issues, analysis of communication methods, examination of security, cost and maintenance issues, etc.

Planning the implementation of the IS architecture (i.e. the set of the functional, organizational and technical models) requires the definition of concrete actions, the evaluation of budgetary requirements, the study of time and organisational constraints, the elaboration on issues of human resources, management and plan coordination, migration and cut-over, etc. In addition, the action plan needs to be examined concerning its risks, its strategic importance, the satisfaction of short-term needs of the organisation and its harmonised integration within the overall evolution of the specific organisation. Finally, follow-up and control procedures should be established in order to monitor and control the cost and to manage the implementation process.

4.3 Team Organisation and Management Participation

The organisational structure of the ISSP process has a twofold objective: clarify management involvement and support teamwork. It is based on the arrangement of project team, team leader and guidance team, following the approach first suggested by Scholtes (1988). The idea is that each project team is formed by, and remains responsible to, the guidance team. The latter is responsible for removing obstacles and providing resources. The guidance team does not manage the project teams; that is the role of the team leaders. The responsibilities of the guidance team are as follows: identification of project goals; preparation of mission statement; determination of project teams; selection and assignment of project teams' leaders and members; determination and provision of other resources needed by the project teams; monitoring the progress of project teams; approval of the results produced by the project teams; accountability for the overall result of the project.

The project teams are made up of people who will do the actual work and are relatively small, i.e. they comprise of 3-4 people. The general characteristic of project teams is that they are mixed in the sense that they include people from the target organisation as well as external experts. Each team is headed by a team leader. Given the need for effective communication and coordination, and the fact that social interaction within a team is a commonly overlooked cause of trouble, the role of the team leader should not be synonymous with "guiding light or chief decision-maker". Rather, a consensus approach to decision-making within the teams should be followed; see also Scholtes (1988). Being team leader means taking on (or nominating another team member to do so) the following responsibilities: facilitator; archivist; manager; and contact; see also Rettig and Simons (1993).

Four different types of teams can be specified: functional; technical; issue-specific; and coordination. The functional teams include business managers from the target organisation while the technical teams include IS managers. The need for issue-specific teams arise when there is a requirement for studying e.g. the impacts of alternative strategic choices to specific parts of the target organisation. Such teams should be made up of high-level executives from the affected parts of the organisation; they can be considered equivalent to the executive workshops proposed by Goldsmith (1991). The coordination team is made up of the project manager and the team leaders of the various project teams.

Finally, the guidance team is made up of the project manager and senior directors of the target organisation. The inclusion of senior business directors (versus to only IS staff) to the guidance team is considered a prerequisite to the successful implementation of the ISSP process.

A graphical representation of the organisational structure is given in Figure 2, while Table 2 gives the correspondence between the ISSP organisational entities and the roles proposed by Friend and Hickling (1987) in the strategic choice approach mentioned in section 3.2.

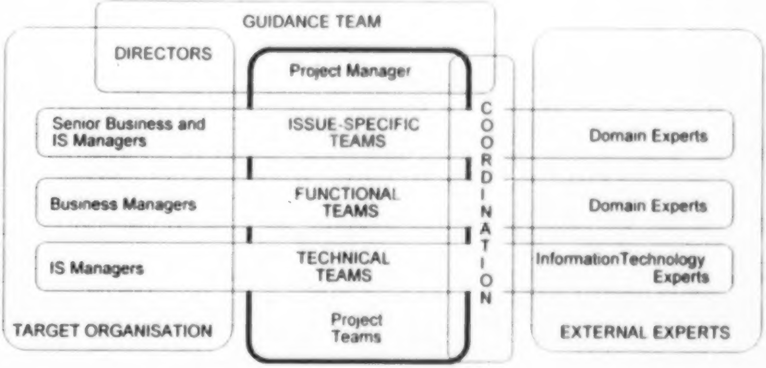


Figure 2. Organizational Structure of IS Strategy Planning Process

4.4 Review, Revise and Approval Procedures

As already mentioned the ISSP process identifies five milestones within each process element: Plan; Approve; Do; Review; Revise; and Evaluate; see also Rettig and Simons (1993). The actual interpretation of the milestones may differ with the phase, the stage and type of module, yet they essentially describe a plan-do-refine approach, punctuated by peer reviews of results.

For each process element there exist the correponding actors. At the level of phases the relevant actor is the guidance team, at the level of stages the relevant actor is the coordination team, while at the module level the corresponding actors are team members. For example, for the case of modules, the milestones are as follows:

Plan. This milestone is passed when the project team leader has prepared a plan for a specific module of activities and submits the plan for review by other members of the project team.

Approve. This milestone is passed when the project team has worked through the initial plan and has reached agreement on a revised plan.

Do. This milestone is passed when the actual effectuation of the activities included within the specific module have been carried out by the project team members.

Review and Revise. This milestone is passed when the results of a module of work have either been approved by the project team members, or revised after the latter communicated any suggestions or concerns. Further iterations of the review-revise cycle may be initiated if needed.

Evaluate. When the module is completed, the team leader and the other project team members evaluate the work, the plan and the process, to look for opportunities for future improvement. Final approval of the module is the responsibility of the coordination team.

A similar quality management procedure is followed for the phase and stage elements of the process but it is monitored by the respective actors.

Table 2. ISSP organisational entities and roles of the strategic choice approach

ISSP Organisational Entities	Roles in Strategic Choice Approach
Guidance team, project manager, team leaders (depending on the decision)	Accountable
Coordination team (project manager, team leaders)	Responsible
Guidance team	Reference
Issue-specific teams	Representative
Directors, Business Managers, IS Managers	Stakeholders

5. CONCLUDING REMARKS

The paper has presented an attempt to develop an information systems strategy planning process (ISSP) towards increasing the practicality and appropriateness of IS strategies within corporate settings. The ISSP process aims towards the linkage of IS strategy with corporate strategy and to the provision of an environment for enforcing the participation and involvement from managerial staff of the target organization. In addition, it attempts to support a structured team approach for the process execution. Elements from the quality management approach are also included, in order to guarantee that review and control activities are explicitly taken into account.

The advantages of the proposed process refer to: the use of a generalised strategy planning approach that can incorporate corporate strategy making issues (and hence be considered familiar to business managers), as well as, include multiple IS strategy development methods; the explicit (and formal) incorporation of structures for management involvement; the support of a group approach, which seems to be the most appropriate for IS strategic planning; and the ability of managers to formally control and review the various deliverables during the whole life-cycle of the planning process.

Some limitations and weaknesses remain. The ISSP process treats planning as an one-shot activity; nevertheless IS planning operations should be on-going management activities and should form integral parts of the dymanic process of defining and monitoring corporate strategy. The ISSP process can be considered as a formalization of what Mintzberg (1993) calls 'strategy programming', i.e. the codification, elaboration and conversion of strategies. Limited formal support is introduced for the 'strategy thinking' aspect, i.e. the synthesizing of qualitative and quantitative data into visions of the appropriate direction to pursue. Finally, although the ISSP process has been applied and has proved its usefulness in several projects during the past two years [see Mentzas (1994b)], significant empirical evidence is still required to refine the approach and further assure its practical applicability.

6. REFERENCES

Ackoff, R. (1981) *Creating the Corporate Future*. Wiley, New York

Brancheu, J.C. and J.C. Wetherbe (1987) Key Issues in Information Systems. *MIS Quarterly*, March, pp. 23-45

Curtis, B., M.I. Kellner and J. Over (1992) Process Modelling. *Communications of the ACM*, Vol. 35, No. 9, September, pp. 75-90.

Deming, W.E. (1982) Quality, Productivity, and Competitive Position, MIT, Cambridge, Mass. 1982.

DeSanctis, G. and R.B Gallupe (1987) A Foundation for the Study of Group Decision Support Systems, *Management Science*, Vol. 33, pp. 589-606.

Earl, M.J. (1989) *Management Strategies for Information Technology*, Prentice-Hall, Hemel Hempstead.

Earl, M.J. (1990) Approaches to Strategic Information Systems Planning: Experience in Twenty-One United Kingdom Companies, *Proceedings of the International Conference on Information Systems*, Copenhagen, pp. 271-277.

Friend, J.K. and A. Hickling (1987) *Planning Under Pressure: The Strategic Choice Approach*, Pergamon Press, Oxford.

Galliers, R.D. (1991) Strategic Information Systems Planning: Myths, Reality and Guidelines for Succesful Implementation, *European Journal of Information Systems*, Vol. 1, No. 1, pp. 55-64.

Galliers, R.D. (1993) Towards a Flexible Information Architecture: Integrating Business Strategies, Information Systems Strategies and Business Process Redesign, *Journal of Information Systems*, Vol. 3, No. 3, July.

Goldsmith, N. (1991) Linking IT Planning to Business Strategy, *Long Range Planning*, Vol. 24, No. 6, pp. 67-77.

Hartog, H.C. and H. Herbert (1986) 1985 Opinion Survey of MIS managers: Key Issues, *MIS Quarterly*, December, pp. 351-361.

Kim, K. and J.E. Michelman (1990) An Examination of Factors for the Strategic Use of Information Systems in the Healthcare Industry, *MIS Quarterly*, June, pp. 201-215.

Kraemer, K.L. and J.L. King (1988) Computer-based Systems for Cooperative Work and Group Decision Making, *ACM Computing Surveys*, Vol. 20, No. 2, June, pp. 115-146.

Kriebel, C. (1968) The Strategic Decision of Computer Systems Planning, *Long Range Planning*, pp. 7-12.

Lederer, A. and V. Sethi (1988) The Implementation of Strategic Information Systems Planning Methodologies, *MIS Quarterly*, September, pp. 445-461.

Lederer, A. and V. Sethi (1992) Meeting the Challenges of Information Systems Planning, *Long Range Planning*, Vol. 25, No. 2, pp. 69-80.

Lee, G.G. and T.G. Gough (1993) An Integrated Framework for Information Systems Planning, in E. Whitley (ed) *Proceedings of the First European Conference on Information Systems*, March 29-30, Henley on Thames, UK, pp. 352-362.

Malone, T. and K. Crowston (1990) What is coordination theory and how can it help design cooperative work systems?, in *CSCW'90, Proceedings of the Conference on Computer-Supported Cooperative Work*, ACM New York.

Mentzas, G.N. (1993) Coordination of Joint Tasks in Organizational Processes, *Journal of Informat Technology*, Vol. 8, pp. 139-150.

Mentzas, G.N. (1994a) Information Systems Strategy for Electronic Banking: The Case of Countries in Reform *International Journal of Information Management* (forthcoming issue).

Mentzas, G.N. (1994b) Strategic Planning for IT in Public Services: External and Internal Perspectives, paper submitted for publication.

Mintzberg, H. (1993) *The Rise and Fall of Strategic Planning*, The Free Press, MacMillan Inc., New York.

Munro, M.C. and S.L. Huff (1985) Information Technology and Corporate Strategy, *Business Quarterly* Summer, pp. 37-42.

Neo, B.S. (1988) Factors facilitating the use of information technology for competitive advantage: an exploratory study, *Information and Management*, Vol. 15, pp. 191-201.

Olle, T.W., J. Hagelstein, et al. (1991) *Information Systems Methodologies*, IFIP, Addison Wesley Publ. Co., New York.

Ormerod, R. (1992) On the Nature of Information Systems Strategy Development, in E. Whitley (ed) *Proceedings of the First European Conference on Information Systems*, March 29-30, Henley on Thames, UK, pp. 455-463.

Porter, M.E. (1980) *Competitive Strategy: Techniques for Analysing Industries and Companies*, The Free Press, New York.

Premkumar, G. and W.R. King (1991) Assessing Strategic Information Systems Planning, *Long Range Planning*, Vol. 24, No 5, pp. 41-58.

Rettig, M. (1990) Software Teams, *Communications of the ACM*, Vol. 33, No. 10, October, pp. 23-27.

Rettig, M. and G. Simons (1993) A Project Planning and Development Process for Small Teams, *Communications of the ACM*, October, Vol. 36, No. 10, pp. 45-55.

Scholtes, P.R. (1988) *The Team Handbook*, Joiner Group, Madison Wis., 1988.

Thompson, J.L. (1993) *Strategic Management*, Chapman and Hall, London.

Ward, J. Griffiphs, P. and P. Whitmore (1990) *Strategic Planning for Information Systems*, John Wiley, New York.

STRATEGIC ASPECTS OF OUTSOURCING

Jochen Schwarze

Institut für Wirtschaftsinformatik, Universität Hannover, Wunstorfer Str. 14, D-30453 Hannover, Germany

Abstract

The increasing application of information and communication technology during the past two decades led to a lot of new concepts. One of these new concepts in information processing and communication (I&C) is outsourcing. Outsourcing means that certain tasks and duties of the information management department are delegated to specialized service companies which are acting as outsourcing vendors. The paper deals with some strategic aspects of outsourcing like outsourcing reasons, economic impact of outsourcing, extension and intensity of outsourcing, choice of an outsourcing partner, relationships between outsourcing company and outsourcing partner, action models for outsourcing, advantages and disadvantages of outsourcing.

1. WHAT IS OUTSOURCING?

1.1 Introductory remarks

The development of information systems and communication during the past two decades is characterized by a sequence of innovations, which sometimes appeared intermittently. These innovations do not only take place in the field of hardware (computers, peripherals, networks etc.) but also in the architecture of information systems and organizational concepts. Typical examples for the latter are

- intercompany integrated information systems like *just-in-time-control*, *electronic cash systems* or *systems of electronic tele-diagnostic or tele-maintenance*,
- delegation of different but related tasks as a whole to a single employee by using integrated task and decision support systems instead of atomistic division of labour.

Another characteristical example is the dislocation of purchasing, processing, storing, transfer and presentation of information to independent service companies. This today is discussed as "outsourcing".

1.2 Definition of outsourcing

Even outsourcing is a new term, it is neither a new business activity nor an innovation of applied computer science. *In general outsourcing means the use of external agents to perform one or more organizational activities*, e. g. purchasing goods or services (cp. Lacity et al. (1993)). Under the headline *make or buy* this is discussed under various aspects in business economics (cp. e. g. Mannel (1981)), usually with respect to efficiency of production, even though *outsourcing* may be applied to nearly every business or administration function.

Even *outsourcing* is a creation of *outside* and *resourcing* which does not imply a restriction to I&C, the term *outsourcing* is only used to discuss the *make-or-buy*-problem with respect to purchasing, processing, storing, transfer and presentation of information.

But even in the I&C domain outsourcing is not a new approach, only a new term, because activities like those discussed as outsourcing, we will find some ten years ago. In the fifties and sixties in many branches companies were using the service centers of hardware suppliers, esp. IBM, for certain tasks of data processing, e. g. statistical evaluation of market analysis, turnover statistics or special tasks of book keeping.

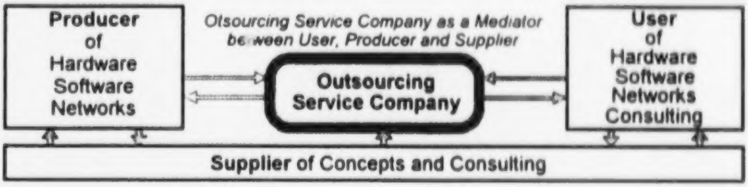
In literature we will find different *definitions of outsourcing*, e.g.:

- Outsourcing is the *delegation of facilities management*, i. e. the complete delegation of the data processing center and all tasks related to the data processing center, to an external service enterprise (cp. e. g. Heinrich (1992, pp. 21)).
- Outsourcing is the *medium- and long-term delegation of secondary functions, esp. the I&C function*, to an external partner (cp. e. g. Szyperski et al. (1993, p. 229)).
- Outsourcing means the use of an external, economically independent I&C service enterprise to perform I&C tasks which have been executed internal so far (Knolmayer (1993, p. 71)). This definition does not include outsourcing to an affiliated company as well as outsourcing of new I&C tasks. In this sence, facilities management is a part of outsourcing.

In the following we will use the term outsourcing for delegation of any type of I&C tasks to an external partner, i. e. we prefer a rather wide definition.

1.3 Business and economic impacts of outsourcing

The delegation of I&C tasks to an external, economically and/or legally independent I&C service company induces new business relations to a new partner or supplier. This new partner supplies certain I&C services and acts as a mediator between I&C user on one side and I&C suppliers of hardware, software, networks and consulting on the other side. This is illustrated in figure 1.


Producer
of
Hardware
Software
Networks
Otsourcing Service Company as a Mediator
between User, Producer and Supplier
Outsourcing
Service Company
User
of
Hardware
Software
Networks
Consulting
Supplier of Concepts and Consulting

Figur 1. Outsourcing service company as a mediator between I&C user and I&C suppliers (cp. Szyperski et al. (1993, p. 230))

Obviously outsourcing will lead to deep and complex changes in the outsourcing company. In production, the *buy* of the *make or buy* decision affects certain steps or activities of production or, in many cases, is related to certain parts of a product. The *buy*-object can easily be separated and is not affecting other processes and activities. But ***in the I&C domain the outsourcing object oftenly is an essential part of the companies I&C system*** (hardware, software etc.) and that means: of the control system. Thus the business impact and importance of outsourcing in the I&C domain mostly is larger than in production, not only with respect to the costs.

The outsourcing market is a growing market with a very high growing rate. The rapidly increasing economic importance of outsourcing will not only be reflected in a large turnover but also in the concentration of market forces on the outsourcing services suppliers because of the dependencies of the outsourcing companies from the outsourcing services suppliers. In 1992 the European outsourcing-market had a volume of 7,6 Billions ECU and will grow up to 16 Billions ECU until 1997. Table 1 shows the 9 largest outsourcing services suppliers in Europe 1992 with their annual turnover (taken from a note in Wirtschaftsinformatik, Nr. 1 1994, p. 91).

Table 1. The 9 largest outsourcing services suppliers in Europe 1992

Supplier	*1992*
EDS	491 ECU
Digital	323 ECU
Debis, Germany	310 ECU
Finsiel, Italy	222 ECU
Cap Gemini Sogeti, France	201 ECU
Datev, Germany	177 ECU
IBM	169 ECU
Sligos, France	161 ECU
GSI, France	159 ECU

In USA a lot of companies makes use of outsourcing to solve I&C problems and to get more efficiency in the I&C area (cp. e. g. Lacity et al. (1993, p. 13)). The new outsourcing services branch makes considerable turnovers as it is shown in table 2.

Table 2. The 6 largest outsourcing vendors in USA 1989 and large outsourcing contracts in USA 1991 (from: Lacity et al. (1993, p. 14))

Company	*Revenue*
EDS	5470 mill. $
IBM	3250 mill. $
Anderson Consulting	1440 mill. $
Comp. Sciences Corp.	1440 mill. $
DEC	1000 mill. $
KMPG Peat Marwick	600 mill. $

Company	*Partner*	*Vol. (mill. $)*
System One	EDS	2.000
Enron	EDS	750
First City	EDS	600
Eastman Kodak	IBM	500
National Car Rental	EDS	500
First Fidelity	EDS	450

1.4 Development of outsourcing

In the seventies and eighties, the use of I&C was influenced by the development of hardware (smaller, cheaper and more efficient), software (standard software, integrated systems) and new concepts (integration, intercompany data processing). The fast I&C development and the induced changes in organization and information processing led to a lot of problems like

- short cycles of innovation and, induced by this, problems of adaption,
- insufficient qualification of employees,
- problems of security,
- high risks of failure,
- reliability problems.

To solve these problems more and more companies are going to outsource I&C tasks with the goal of optimizing benefits.

The development of I&C to outsourcing is characterized by the following phases:

- In the "classical" data processing department or computer center all I&C activities were concentrated. It was operating as a closed shop or black box and not opened for users.
- The development of hardware and software led to decentralization of hardware (PCs, work stations, networks) and to personal computing or individual data processing. This was accompanied by an increasing demand of support and service to the users of decentralized I&C.
- The high degree of decentralization and the fast development of I&C technologies in connection with increasing complexity (and sometimes decreasing reliability) and a lag in qualification of employees induces outsourcing.

2 REASONS FOR OUTSOURCING

In literatur we can find a wide spectrum of reasons to outsource I&C tasks. The following survey gives an overview but could not lay claim to completeness (cp. to the reasons of outsourcing e. g. Heinrich (1992), Knolmayer (1993), Lacity et al. (1993), Lang (1992), Szyperski et al. (1993)).

Internal motivation to outsourcing comes from

- strategic reasons: increasing orientation to processes and projects and turning away from orientation to functions, general trend to outsource service functions, concentration to the core business of a company and in connection with this the delegation of secondary functions to service companies: "do what you can best - outsource the rest",
- personnel reasons: unsufficient qualifications of users, low qualifications of I&C employees or for Lack of I&C specialists;
- economic and organizational reasons: demand on rationalization efforts, high I&C costs, problems of authority and competence between functional departments and the I&C department;
- reasons from unsufficient I&C in the past: deficient I&C in the past, increasing complexity in I&C, short cycles of innovation in hard- and software, increasing problems of hardware and network management, problems of security.

In a study of Lacity et al. (1993, pp. 198) the following six reasons were pointed out as the main motivation for outsourcing:

- Reaction to the efficiency imperative
- The need to require resources
- Reaction to the bandwagon
- Reduce uncertainty
- Eliminate a troublesome function
- Enhance credibility

External motivation to outsourcing comes from the market or from the services offered by the outsourcing vendors:

- optimal utilization of specialized I&C employees,
- high qualified staff,
- input of modern hard- and software,
- short cycles of innovation in hardware, software and I&C concepts,
- reserve capacity,
- optimal capacity control,
- high potential in automatization of system operating and network operating,
- short reopening time in any case of damage.

3 OBJECT OF OUTSOURCING

3.1 Basic ideas about the I&C tasks which can be outsourced

In practice there is a wide range of different ousourcing contracts and solutions. They are reaching from temporal limited delegation of a single I&C task like data collection or scanning questionaries and the statistical evaluation of the data gathered by the questionaries up to a nearly complete outsourcing of the I&C department. A total outsourcing of I&C is impossible because information are arising in the company as well as the results of information processing were needed in the company. At least the interfaces must be left in the company. This is a characteristic of outsourcing and implies a more or less integration of the outsourcing vendor into the I&C processes of the outsourcing company.

Figure 3 illustrates how outsourcing resp. the outsourcing vendor is integrated in the overall I&C process of the company.

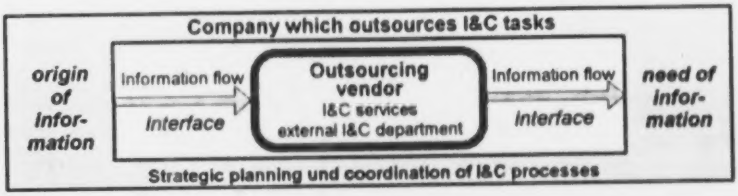

Company which outsources I&C tasks
origin of infor-mation
Information flow
Interface
Outsourcing vendor
I&C services
external I&C department
Information flow
Interface
need of infor-mation
Strategic planning und coordination of I&C processes

Figure 3. Intergration of an outsourcing vendor in the companies information process

Thus from the companies view and the companies overall I&C process we have natural limits of outsourcing. Limits we will also find looking to the I&C tasks in the company. The development and updating of an information strategy and the design and planning of basic or strategic concepts of I&C processes should be left inside the company because of there vital importance.

3.2 Width of outsourcing

The strategic planning of outsourcing has to determine, which functions or tasks should be delegated to an outsourcing vendor.

From the viewpoint of the companies divisions or departments outsourcing domains may be e. g. from:

- purchasing,
- materials management,
- production,
- selling,
- finance,
- accounting,
- research & development,
- project management.

From the viewpoint of the I&C department we can distinguish between:

- system development,
- system operating,
- network management,
- computer center,
- user support and service,
- maintenance,
- programming,
- data collection.

Looking to the duties of an information management (cp. e. g. Schwarze (1990) and (1993)) the strategic outsourcing decisions should reflect the following:

Strategic duties are not or only partly suitable for outsourcing:

- *Planning and design of information infrastructure* has long term impacts and plays an important role in the general strategic planing. Thus outsourcing will mainly be restricted to consulting. The fundamental decisions and planning duties must be done by the (top) management.
- *Management of technological innovation* requires a high level of specialized knowledge, thus outsourcing will bring benefits.
- *Design of I&C systems and organizational concepts* touches the whole organization and so only parts of these tasks are suitable for outsourcing.
- The *development of an information strategy* is a duty of the top management and should not be delegated to an outsourcing vendor.

The tasks of *system design and realization* could be outsourced in a high degree, concerning

- analysis of the present state,
- requirements analysis and specification,
- software development,
- hardware configuration,
- management of I&C projects,
- purchasing software,
- design of data bases,
- design of security concepts.

Outsourcing is also rather easiliy possible for nearly all *tasks of operating I&C systems*:

- service for decentralized hardware,
- hardware maintenance,
- network management,
- computer center,
- user support,
- data management,
- security management,
- disaster management,
- I&C revision,
- accounting.

The decision about outsourcing width depends on the following criteria:

- strategic role for the company,
- security requirements,
- requirements to the level and quality of realizing I&C tasks,
- degree of automation and integration.

3.3 Depth of outsourcing

Outsourcing depth is refering to the question, if a special I&C task should be delegated to an outsourcing vendor in whole or in part. Most tasks of operating I&C systems, like transaction processing, production service and utility processing, could be undertaken by an ousourcing vendor. Conception and design of I&C systems, duties of security and controlling must be left partially inside the company. But today sometimes outsourcing concerns even the area of I&C management.

3.4 Special outsourcing domains

Outsourcing is practicized somtimes to spezial domains, e. g. to support individual information processing by

- provision of hard- and software,
- installation and management of networks,
- maintenance of hard- and software,
- training and support when using new software,
- development of new applications and
- user service.

A particular case is every system of intercompany I&C processing like "just-in-time" or "electronic banking". These systems inevitably lead to outsourcing.

3.5 Support systems for the outsourcing decision

Planning and preparing outsourcing of I&C tasks and I&C facilities is a very complex and partly fuzzy problem. So obviously managers are looking for decision support, for which there are different techniques and methods. Knolmayer (1993) discusses seven different decision support approaches and their application to the outsourcing problem:

- portfolio analysis,
- check list,
- arguments balance,
- benefit value analysis,
- cluster analysis,
- complete enumeration,
- different modells of mathematical optimization.

Most of these approaches are difficult to apply because it is complicated or nearly impossible to get the information necessary for applying the models.

4 CHOICE OF AN OUTSOURCING PARTNER

The choice of an outsourcing partner ist difficult, because the outsourcing vendor doesn't act as an usual supplier becaus his service is - depending on the outsourcing concept - more or less integrated in the operational procedure of the company. This is caused by the role of information and I&C systems in the internal processes and organization. So outsourcing does not lead to the usual relationship between customer and supplier. Outsorcing is more, it is a special type of partnership.

The following criteria may be helpful to support the choice of an outsourcing partner:

- level of know-how,
- experiences in the interesting I&C domains,
- neutrality with respect to hard- and software suppliers,
- service and support in the phase of conversion and operation,
- personnel and technical resources,
- mid term and long term capabilities to extend the partnership,
- strategic objects of the outsourcing partner, compatibility of his objectives and his strategy with the corporate objectives and corporate strategy,
- competitive ability, innovation capability and economic power of the outsourcing partner,
- degree and level of specialization in those fields of I&C which should be outsourced,
- service level,
- price.

The relationship between the outsourcing company and the outsourcing partner may be of different types, e. g. the outsourcing partner could be

- an affiliated company, which is (1) only active for the parent company or (2) also for other companies,
- a company, independent in law and economically.
- Other solutions are
- the outsourcing partner acquires the computer center of the outsourcing company,
- the outsourcing leads to a reduction of I&C resources and I&C employees,
- outsourcing is only realized to special I&C domains,
- the outsourcing partner is a specialized I&C service company for a certain economic or business sector or branch.

4.6 An action model for the choice of an outsourcing partner

Outsourcing induces long term business ties and thus a systematic action model for the choice of an outsourcing partner is necessary. Two proposals are shown in figures 4 and 5. Figure 4 shows the phases of planning and preparing an outsourcing decision.

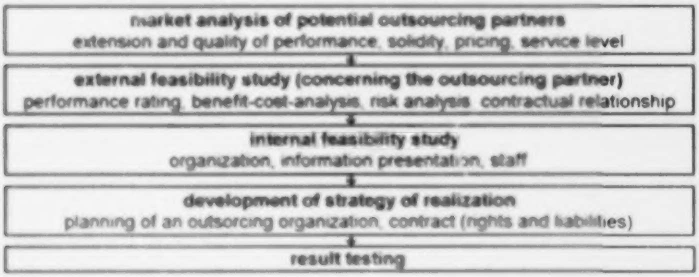

market analysis of potential outsourcing partners
extension and quality of performance, solidity, pricing, service level
external feasibility study (concerning the outsourcing partner)
performance rating, benefit-cost-analysis, risk analysis, contractual relationship
internal feasibility study
organization, information presentation, staff
development of strategy of realization
planning of an outsorcing organization, contract (rights and liabilities)
result testing

Figure 4. Action model for planning outsourcing (cp. Lang (1992, p. 74))

Figure 5 is directly oriented to the choice of an outsourcing partner. Both models shown in figure 4 and 5 may be modified with respect to the individual situation.

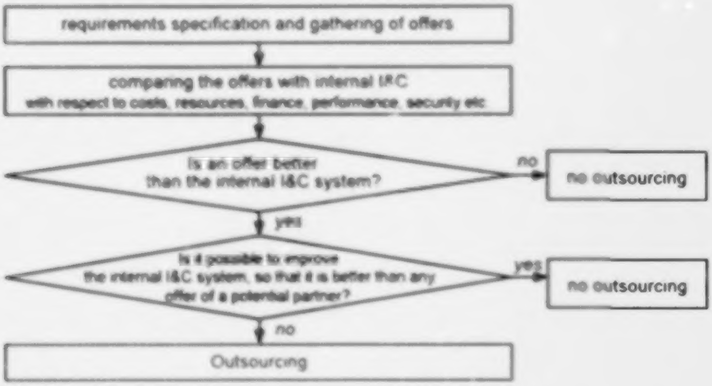

requirements specification and gathering of offers
comparing the offers with internal I&C
with respect to costs, resources, finance, performance, security etc
Is an offer better
than the internal I&C system?
no
no outsourcing
yes
Is it possible to improve
the internal I&C system, so that it is better than any
offer of a potential partner?
yes
no outsourcing
no
Outsourcing

Figure 5. Action model for the choice of an outsourcing partner (cp. Lacity et al. (1993, p. 231))

5 ORGANIZATION OF OUTSOURCING

The result of outsourcing is a cooperation between two partners, the outsourcing company and the outsourcing vendor. To ensure high efficiency of this cooperation, the operation of outsourcing requires an adequate organization. The management of both partners is controling the outsourcing. This outsourcing management or parts of it may be delegated to a control and coordination committee which is responsible for the whole outsourcing cooperation. The outsourcing process itself resp. the outsourcing activities then could be regarded as something like an intersection between the two partners.

As I&C processes play an essential role in the work flow resp. business processes, it is necessary, to install a quality control to supervise the outsourcing activities.

Based on the outlined ideas Figure 6 shows how outsourcing may be organized. The arrows are illustrating flows of information, instructions and work.

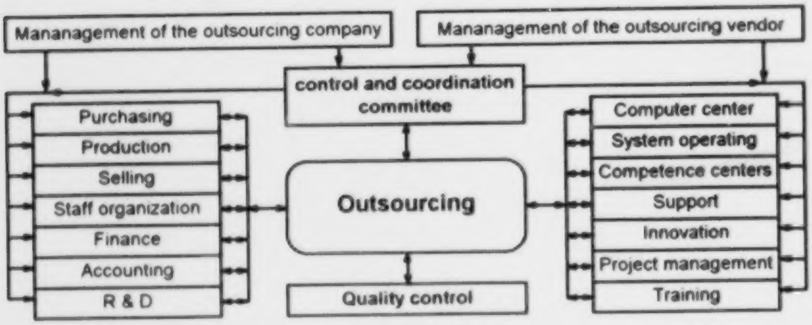

Mananagement of the outsourcing company
Mananagement of the outsourcing vendor
control and coordination committee
Purchasing
Production
Selling
Staff organization
Finance
Accounting
R & D
Outsourcing
Quality control
Computer center
System operating
Competence centers
Support
Innovation
Project management
Training

Figure 6. Organization of outsourcing

6 ADVANTAGES AND DISADVANTAGES OF OUTSOURCING

Outsourcing is entailed with different advantages and disadvantages. The following catalogue gives an overview (cp. e. g. Heinrich (1993), Heinrich (1993a, p. 43), Knolmayer (1994), Lang (1992, pp. 74-80), Sommerlad (1993, p. 48)):

advantages
- reduction of the complexity of I&C processes,
- acces to innovations,
- acces to high competence and know how,
- access to high standards in hard- and software,
- access to high processing and storage capacity,
- reduction of the danger of a bottleneck in I&C processes,
- reduction of technical and personnel risks,
- concentration to the primary functions or core business,
- flexibility,
- better service,
- risk transfer,
- cost reduction by economies of scale, fewer training, fewer measures in security and data protection, no capacity reserves for processing peaks;

disadvantages
- dependence from the outsourcing partner,
- problems of coordination and control,
- loss of competence and know how,
- risk of bad performance of the outsourcing partner,
- frictional loss between internal and external employees,
- problems of acceptance in the functional departments,
- costs of transaction, data transmission, reorganization, accounting, coordination and control, administration.

7 FINAL REMARKS

Outsourcing induces a lot of changes in the whole domain of information management and related areas. Thus an outsourcing strategy is needed which includes an action model for the choice of an outsourcing partner as well as criteria and concepts for outsourcing width and depth. In the process of current operations a new position or department is necessary to coordinate and control the outsourcing process. This includes all aspects of cooperation with the outsourcing partner.

Looking to the advantages and disadvantages one can suppose that in many cases it would be difficult (or nearly impossible) to decide about outsourcing by a cost-benefit-analysis. Most benefits are qualitative and could not be used for calculation of profitability. But one can assume that outsourcing gives a lot of benefits especially because of the access to competence, know how, actual and modern hard- and software and reserve capacity. Thus outsourcing would be a way to solve problems in the I&C domain of a company.

REFERENCES

Heinrich, W. (1992): Outsourcing, Modewort oder neues strategisches Konzept? In: Heinrich, W. (Ed.): Outsourcing, Modelle - Strategie - Praxis. Bergheim 1992, pp. 11-54.

Heinrich, W. (1993): Wann es sich lohnt, DV-Aufgaben auszulagern. In: Business Computing vol. 6, pp. 42-45.

Heinrich, W. (1993a): SAP-Outsourcing: Festpreise trügen. In: Online vol. 11, pp. 59-60.

Knolmayer, G. (1993): Modelle zur Unterstützung von Outsourcing-Entscheidungen. In: Kurbel, K. (Ed.): Wirtschaftsinformatik '93. Heidelberg 1993, pp. 70-83.

Knolmayer, G. (1994): Der Fremdbezug von Information-Center-Leistungen. In: Information Management vol. 1, pp. 54-60.

Lacity, M.C.; Hirschheim, R. (1993): Information Systems Outsourcing. Chichester et al.

Lang, M. (1992): Ein Votum für partnerschaftliche Kooperation. In: Heinrich, W. (Ed.): Outsourcing, Modelle - Strategie - Praxis. Bergheim 1992, pp. 55-85.

Männel, W. (1981): Die Wahl zwischen Eigenfertigung und Fremdbezug. Stuttgart, 2. Aufl.

Schwarze, J. (1990): Betriebswirtschaftliche Aufgaben und Bedeutung des Informationsmanagements. In: Wirtschaftsinformatik vol. 32, pp. 104-115.

Schwarze, J. (1993): Qualifizierungskonzepte für das Informationsmanagement. In: Scheer, A.-W. (Ed.): Handbuch Informationsmanagement. Wiesbaden 1993, pp. 633-653.

Somerlad, K. (1993): Der Outsourcing-Vertrag in der EDV. In: Business Computing vol. 8, pp. 48-50.

Szyperski, N.; Schmitz, P.; Kronen, J. (1993): Outsourcing: Profil und Markt einer Diestleistung für Unternehmen auf dem Weg zur strategischen Zentrierung. In: Wirtschaftsinformatik vol. 35, pp. 228-240.

Applications

IDA: AN INTELLIGENT INFORMATION SYSTEM FOR HETEROGENEOUS DATA EXPLORATION

Maria Luisa Damiani

ELDA Milano S.r.L.,Via Pirelli 27, 20124, Milano, Italy

Abstract

The paper presents an industrial application of an object based knowledge representation system relying on description logics. The knowledge representation language is used as core component of IDA, a complex information system for data exploration support, developed in the Esprit project AIMS. In this paper we describe the motivations, the architecture of IDA and the methodological guidelines for building an exploration-oriented application in the IDA framework .

1. INTRODUCTION

A challenging application for intelligent information systems is to aid the inspection and interpretation of large amounts of coarse data. Applications of this kind are for example: analysis of data generated by scientific experiments (Ioannidis and Linvy, 1992); 'data mining' for knowledge discovery in large databases (Brachman and Halper, 1992); inspection of corporate business information (Damiani and Bottarelli, 1990). A distinguishing feature of these applications is the high degree of indefiniteness of the search process. In fact in most cases the information which is actually "interesting" is not known in advance but results from a process of data *exploration*.

In the AIMS project[1], it has been developed an information management system called IDA to support the exploration of knowledge, database and multimedia information. Goal of the system is to provide a very flexible environment for intelligently classifying and navigating related information possibly of different nature. IDA results from the integration of different technologies: knowledge management, database, multimedia and visual programming. The core of the system is the so called Intelligent Repository which contains the conceptual representation of the information base content. Such a representation is shown and accessed by the end user through an intuitive graphical interface constituing the IDA Desktop. The other fundamental modules are: the Multimedia Object Manager (MOM) for the physical management of unstructured entities and the Database Linker for transparently accessing a preexisting relational database through the Intelligent Repository.

Because of the need of a very expressive description of the Repository's content it was used a representation tool based on description logics. Unfortunately this paradigm, though extensively studied from a theoretical point of view, is still little applied in real applications. As a consequence it lacks a methodological base which can help developers to decide which representation constructs are most suitable in which cases. In this paper we present some methodological hints for applying description logics to the development of just a specific class of applications, the IDA exploration-oriented applications. For grounding the presentation on a concrete application, we refer to the case study developed in the course of the AIMS project, concerning technical documentation management.

---

[1] The work reported in the paper has been partially supported by the EEC under the ESPRIT Project 5210, AIMS. Partecipants are: Datamont (I), Quinary (I), Technische Universitat Berlin (D), Universidad Pais Vasco (S), Non Standard Logics (F), ONERA-CERT (F).

The paper is structured as follows: first we describe some basic features of the exploration activity; next we introduce the baseline of the IDA approach and the general architecture; next we present some methodological guidelines; finally we describe how the exploration of multimedia information is supported in IDA.

2. THE EXPLORATION ACTIVITY AND THE IDA APPROACH

2.1 Exploration Scenario

Throughout the paper we take the area of technical documentation management as sample application context for the IDA system. In organizations like engineering or manufacturing there is a great variety of documents and data such as drawings, technical specifications, measurement files. Documents are generated at different stages, by distinct departments and are generally accessed by professionals such as engineers or technicians. In this setting, information access is often performed by data exploration. In our case study, a sample activity based on data exploration is the diagnosis of critical faults in a plant for electric energy production. The critical step of the overall activity, which can take a few weeks, is indeed finding out the information which is relevant to detect the possible causes of the fault. In fact it does not exist any predefined path carrying to the useful information and the technician proceeds generally inspecting different data sets and reasoning on them. Though quite simple, the example points out a number of features of the search process:

- search is content driven;
- search is a process not an atomic operation. It expands in time and has a context. The context is constituted by the relevant information progressively collected;
- the atomic operations of search are: querying and stepwise navigation;
- queries are ad hoc and not pre-packaged.

The goal of IDA is to support this abstraction in case information consists of both structured and unstructured data. For sake of simplicity, unstructured data is limited to images.

2.2 The IDA Approach

In our view the key issue for supporting exploration is to provide the user with an understandable representation of the heterogeneous information base structure and content, i.e. of the conceptual schema.

By definition, conceptual schemas capture the structure and constraints of the data that are recorded in a database so that only valid data is accepted for storage. In conventional DBMS the knowledge of the schema is generally not relevant for the end user. In fact schemas are either small and thus the user can easily memorize the database structure or the interaction is through pre-packaged queries. Conversely when data exploration is performed user interactions are mostly in the form of ad hoc queries. In combination with the size and complexity of the information bases this makes the knowledge of the schema much more valuable than in a conventional DBMS (Ioannidis and Linvy, 1992). In IDA the conceptual schema is the central piece of information which provides a common foundation for all types of interaction between the users and the system. That introduces a number of requirements on the data model in which the schema has to be expressed:

- the data model must have high expressive power. The primitives of the data model are to be closely related to notions that users currently use.

• the schemas in the data model must have a succinct representation so that they are easily understood by users. Intuitive interaction metaphors supported by user-friendly interfaces are a prime requirement.

The approach proposed in IDA is to use an object-based knowledge representation language relying on description logics for representing the conceptual schema. These language, also known in literature as terminological languages all descend from the KL-ONE system (Brachman and Shmoltze, 1985) (Patel-Schneider et al., 1990) and provide a sound and formal basis to the paradigm of semantic nets. The specific language we have used is called BACK (Von Luck et al., 1987) (Hoppe et al., 1993).

2.3 The BACK Language

The BACK language and in general all description languages, distinguishes between *terminological* (TBOX) and *assertional* (ABOX) knowledge. Terminological knowledge captures the intensional aspects of a domain and consists of a set of *concepts* and *roles* organized in *subsumption taxonomies*. Concepts are used to collect relevant entities of a domain; roles are used to specify properties of concepts and thus model relationships between concepts. A distinguished feature of the language is the possibility of specifying definitional properties for both concepts and roles (i.e. properties necessary and sufficient for an individual/couple to be a concept/role instance). Concepts and roles which express definitional properties are called respectively *defined concepts* and *defined roles*. For example the declaration:

mother:= woman and atleast(1, child)

specifies a necessary and sufficient condition for a woman being a mother. Conversely concept (roles) for which it is not possible to give a semantic characterization in terms of other concepts (roles) are called *primitive*. For example, the declaration:

person:<mammal

specifies that a person is necessarely a mammal, but it does not provide sufficient conditions to establish when an instance of the concept 'mammal' is also an instance of the concept 'person'.

The basic inference is called classification. It places automatically concepts in the net of concept descriptions depending on the specified properties, while instances are automatically completed with the most specific description. Both concepts and roles can be composed in complex terms by applying a number of composition operators: restrictions on values and cardinality of roles; conjunction of concepts and roles; transitive closure, inverse and composition of roles. Disjointeness of concepts can be stated as a constraint. It is possible to define simple rules called I-links on concepts. Finally in BACK it is possible to declare functional roles, that is roles in which the value of role fillers is computed by an external function. For an exhaustive description of BACK syntax and semantics - not included in this paper - refer to (Hoppe et al. 1993).

2.4 Architecture

BACK is used for building the central component of IDA called *Intelligent Repository* which contains the conceptual descriptions of both the structured and unstructured entities in the domain. The architecture of the system is depicted in figure 1. A more extended description is given in (Bertino et al., 1993).

Each entity in the domain is described by a BACK object. Actual entities can be stored in different places. Structured data can be stored in the repository itself, otherwise in an external and pre-existing database. In the latter case the integration strategy is based on the schema mapping approach. The idea is to abstract and represent in BACK the content of the database (or part of it) while maintaining in a "mapping" description the correspondence between the knowledge base structures and the database relations (Menaglio et al., 1993). The *Database Linker* is in charge of interpreting the queries, dispatching requests to the database and generating the answers in terms of instances which are entered into the knowledge base.

As concerns unstructured data, i.e. pictures the idea is to use BACK for modelling the conceptual and spatial description of images and to build a *Multimedia Object Manager (MOM)* for their physical management (Bertino et al., 1992). The MOM associates to each conceptual object a so called physical object which specifies low level attributes of pictures and the operations which can be called on them. The operations can also be performed by external applications interoperating with the MOM. In the latter case the approach used for coupling the system with the application tools is based on the Operational Mapping (Bertino et al., 1993).The idea of the strategy is to abstract a common protocol of operations for interacting homogenously with application tools. Therefore the integration protocol consists of a minimal set of messages to which each encapsulated application must respond.

The Intelligent Repository is accessed by the user through a visual interface consisting of a number of facilities for supporting explorative search and information updating. The interface constitutes the *IDA Desktop* . A valuable feature of the IDA Desktop is that retrieval can be performed by interleaving querying and hypermedia-like navigation across the net of structured and unstructured objects. The basic facilities of the interface are: a query-by-form-like tool for querying the knowledge base augmented with images (Query-by-hyperform); a tool for navigating a collection of objects traversing in a stepwise manner the relationships which logically link objects (Navigator); a tool for creating and modifying links among objects (Creator).

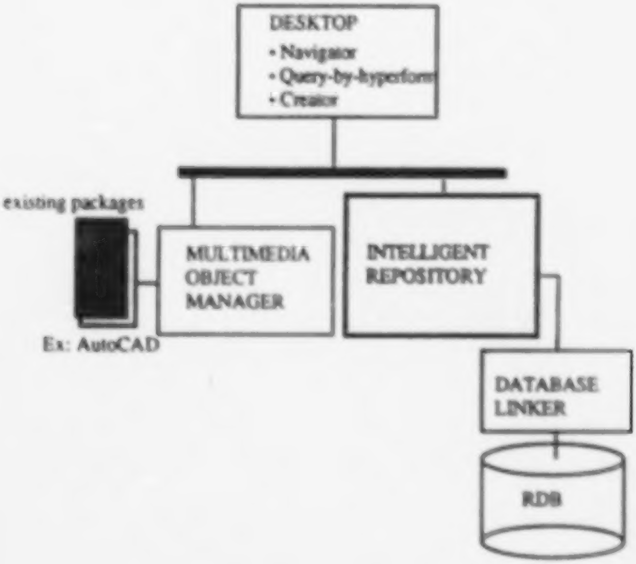


Figure 1. Architecture of the IDA system.

3 THE REPOSITORY: METHODOLOGICAL GUIDELINES

Developing an application in IDA means essentially to build a repository which supports the exploration of a specific data set. A distinguishing feature of the process of knowledge base construction in IDA is that it exists a tight interconnection between the way information is structured and the way it is presented to the user. We remind that the user is supposed to be a domain expert thus someone who is not at all concerned with the deep technicalities of the system and strongly demand a simple to use system. For that reason, the effectiveness of presentation greatly depend on the conceptual clarity of the domain model. For example the user unlikely appreciates objects he/she is not familiar with, which have been possibly introduced only for representation convenience at knowledge base level. In that case it might be useful to distinguish in the specification what is domain dependent and what instead implementation dependent and have a different management of the entities at desktop level. The problem could be more elegantly approached by allowing the definition of some form of external schema for the knowledge base likewise in databases. In that case the external schema could be to some extent tailored on the user needs and the knowledge base could be developed almost independentely from presentation aspects. A proposal in that direction is described in (Bertino et al. 1993). However, even in the latter case it remains the problem of organizing the schema in a way understandable to the user. The issue is challenging. The methodology investigated does not solve the problem, nevertheless it offers some hints which can clarify the way the knowledge representation language is used for exploration tasks. In essence the idea is to exploit the representation and inferencial capabilities of the description language to provide a number of facilities called: *automatic indexing, object decomposition, automatic object linking*.

3.1 Automatic Indexing

The rationale is that a fine grained classification of objects simplifies search but it can be too expensive if performed manually especially when there are many objects and many collections in which the objects can be entered. The so called automatic indexing is a facility which makes the system recognize the class/es an object belongs to on the basis of the object properties. Using a description language like BACK, the automatic indexing is given almost for grant.

However there are two cases to be distinguished which are to be differently represented in BACK: the first is when objects are described by properties which are *all* necessary and sufficient. In that case the automatic indexing onto a class is obtained simply representing the class as a BACK *defined concept*. As an example consider the following informal description:

A business_offer is a document sent by some supplier and whose subject is 'offer'

The properties of the business_offer are both necessary and sufficient. When a generic document is sent by someone which is known to be a supplier and the subject is an offer, the document is classified as a business_offer. In BACK it can be expressed as:

business_offer:= document and all(sender, supplier) and has_subject:'offer'

The second case is when only few (but not all) properties are sufficient. In that case the necessary properties are expressed introducing a primitive concept while the sufficient conditions are represented by rules (called I-links in BACK). As an example consider a simplified typology of fault causes. Each cause can be described by a code and by a string. For example "Lack of maintenance" is given the internal code 2010 and a string "carenza manutenzione". The same string can be used to describe different faults. Conversely the code identifies univocaly the fault. In BACK it can be expressed by introducing a primitive concept and a I-link as follows:

```
lack_of_maintenance:< cause and has_code: 2010 and has_label: "carenza manutenzione"
has_code: 2010 => lack_of maintenance
```

3.2 Object Decomposition

Often an object can be identified through one or more of its parts. For instance a technical drawing is a composite picture made of significant parts which can be inspected during the search process. In general even for exploration purposes it seems important to represent the association commonly known as *part_of* which relates an object to its parts.

In BACK the part_of relation is not a native construct of the language. For sake of simplicity, instead of extending the language, we have preferred an approach in which we try to represent the meaning of the relation directly in BACK (Damiani 1993). In that respect a relation is considered of part_of type when it constitutes a partial order over a set of objects, therefore it is antiriflexive, asymmetric and transitive. We call the set of objects on which the partial order is defined the *scope* of the relation. The partial order has a *top element* which consists of the entire object. By definition, the top element cannot be part of any other object in the set. For example, the scope of the relation "part_of_engine" consists of the entire engine (top element) and of the elements composing it.

The above model of part_of can be expressed to a great extent in BACK. The *scope* of a part_of relation is represented by a BACK concept. Some *basic* roles are defined for the concept which are unique for each part_of relation: the role which relates a part to its immediate father and its inverse. The transitive property of the relation is expressed by two additional roles respectively for the ancestor and discendent parts which are defined as transitive closures of the basic roles. An example can clarify the approach. Consider the parts of an engine. The set of parts constitute the scope of the relation for which the basic roles are specified.

```
engine_part: < anything                                       ; scope
part_of _engine: <domain (engine_part) and range(engine_part)  ; basic role
consists_of_part: = INV(part_of_engine)                        ; basic role
entire_engine:< engine_part and no (part_of_engine)            ; top element
ancestor_part:= TRANS (part_of_engine)                         ; transitive closure
descendent_part:= TRANS (consist_of_part)                      ; transitive closure
```

Note that the property of asimmetricity of the part_of relation cannot be expressed in BACK because it is not possible to prevent cycles among objects. A number of additional properties allow different types of part_of to be defined.We extend the classification presented in (Von Kim and Bertino, 1989) distinguishing among :

- *physical* or *logical* part_of (an object can be necessarely part of another object or not);
- *exclusive* or *shared* part_of (an object can be part of only one or more objects) ;
- *structured or unstructured part_of* (an object can consist of parts which can be homogeneous or not).

Unfortunately not all the forms of part_of can be fully represented in BACK. Note in particular that the physical part_of cannot be properly modelled in BACK. In fact an object which is defined to be necessarely part of another object is not invalidated or removed when the object it belongs to (the object father) is deleted. That is due to fact that in description systems the delete operation is typically monotonic, that is the knowledge base remains consistent when an object is deleted. In practice the consistency is preserved because the system replaces the given object with an "unknown" entity. As a consequence for representing the part_of relations which are not fully supported by BACK it is necessary to add a procedural part.

3.3 Derived Links

One of the atomic operations performed during exploration is the stepwise traversing of associations. In this respect the knowledge base provides a semantic foundation to the navigational facilities which have been developed at Desktop level. However in order to be really helpful, navigation must be very flexible:

- the user can desire to traverse in whatever direction the associations;
- the user can desire to move straightforward from an object to another object, skipping the intermediate steps;
- more intricate is the net and more interdependent are the associations, more difficult is maintaining its consistency when something change.

To fulfill the navigation requirements, it has been introduced the notion of *Derived Links*. Derived Links are used for entering into the knowledge base inverse associations and path shortcut. Derived Links are simply represented in BACK by *defined roles*. By composing and inverting arcs, even arcs representing Derived Links, the graph can be tailored upon the navigational needs. The great benefit of using BACK is that the consistency of the net is preserved by the system when some change occurs, because of the addition or updating of instances. In fact the *defined role* is such that the system takes care of checking and propagating the effects of changes. An example of how a very simple net can be configured by the addition of a Derived Link is the following. Suppose you have entered this description into the knowledge base (for sake of simplicity the description is expressed in natural language instead of BACK):

dossiers contains documents
documents refer necessarely to a plant
documents can be about faults
failures concerns necessarely a plant

The graph representing the description is in figure.2. In this simple graphic notation circles stand for concepts and labelled arrows for roles. For a better comprehension of the picture concept forming operators are not represented.

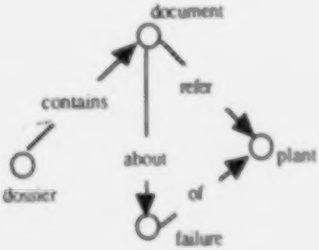


Figure 2. Graphical representation of BACK description

The paths which can be traversed by the user at Desktop level are the paths of the directed graph. In the example of figure 2, the user can select a dossier, move to the documents inside it and then move to the details of the plant a document refers to. Such type of graph can be easily configured by adding arcs and thus introducing alternative paths. The sample arc connecting "failure" to "dossier" and expressing the relation between a failure and the dossier which contains documents about that failure can be expressed in BACK as follows:

SampleArc:= COMP (INV (about), INV(contains))

In such a way, the graph can be traversed in a different manner according to the scenario in which the user whishes to operate.

4 THE EXPLORATION OF MULTIMEDIA INFORMATION

We have presented so far some guidelines for building the IDA repositories to support the exploration task. We go now a step further and see how the strategy is applied for the exploration of multimedia information, consisting of knowledge and images. The key points of the approach to the management of images are:
- representation of images
- user navigation across knowledge and images

4.1 Representation of Images

An image is assumed consisting of two dimensions, respectively conceptual and physical. At conceptual level, the image is described as an object representing some other object in the domain. In general, the meaning of an image is mostly domain dependent.The physical dimension is about the physical properties of images, such as the image format. The conceptual description is contained in the Repository while the physical characterization is stored in the Multimedia Object Manager. The two worlds relates through the Desktop component.

As concerns the conceptual description, we focus on here, the basic idea is that the only aspect which can be generalized to a certain extent is the structural model. For that purpose, it has been defined a skeleton of the image structure which can be filled in and specialized for the different types of images. Accordingly pictures are represented by a specific concept called GrOb (Graphical Object). A GrOb expresses the following information:
- *represented entities:* a GrOb may represent some domain entity, that is some other conceptual object. This relationship may be modelled by the role *represent* between the concept GrOb and the concept standing for the domain entity ;
- *parts*: a picture consists of portions which in turn can contain other portions. The portions are Graphical entities too. A part_of relation relates the portions in a picture. This form of part_of has the following properties: physical (each portion is necessarely part of another one); exclusive (a portion is part of at most one other object); unstructured (the set of portions is homogeneous);
- *spatial relations:* simple spatial relations such as up, down, right, left can be defined between portions.

Because images are represented by BACK concepts, it makes sense to use automatic indexing to classify pictures on the basis of what they represent or contain. Consider that the reasoning process uses both textual and pictorial information. An example in which this form of combined reasoning is performed is the following. Assume "dangerous plant" is defined as:

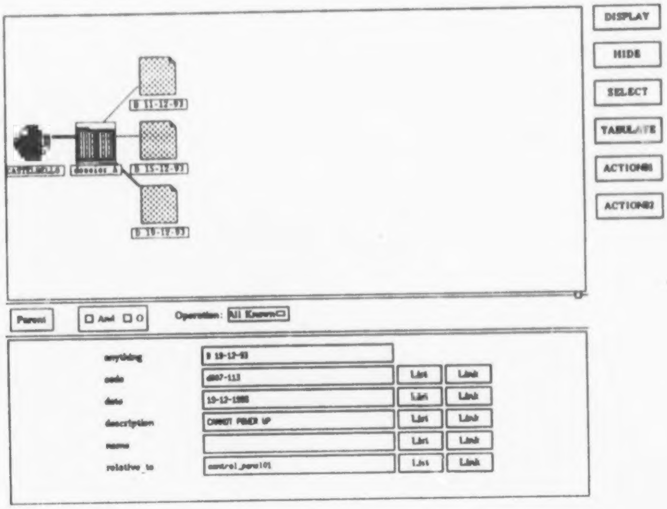
"a plant represented by a map which contains a portion representing a container of explosive near a portion representing a source of heat".

Each time a new map is entered, the system checks whether the conceptual entities verify the spatial relationship of being near. In such a case the systems infers that a dangerous situation exist and thus the plant is classified as "dangerous plant".

4.2 User Navigation

The user perceives the knowledge base as a graph which can be traversed through the Desktop functionalities. As previously said the graph can be to a certain extent tailored on used needs and alternative paths can be added to the graph entering Derived Links into the knowledge base. The graph is used by the Desktop for guiding the user in an intuitive way within the knowledge base. Although the desktop provides three different ways for accessing the knowledge base, i.e. stepwise navigation, querying and creation/updating, the interaction style is rather homogeneous. Each object (i.e. BACK object) is displayed through a form . Forms can be linked through *link buttons* to other forms to graphically represent relationships among objects. The graph which is progressively traversed by the user while navigating the knowledge base or composing the query or the description is displayed by the so called *iconic browser* (Fig.3). In the browser window each object is represented by an icon, identical for all the elements of a concept. The user can position himself/herself on each node of the graph. Each navigation step is visualized by adding an arc to the iconic browser. Note that the user can move in two main directions, "horizontal" i.e. following the associations among objects, "vertical" when moving from the generic description to a more specific concept through the subsumption hierarchy. However the latter direction makes sense only when composing a query or a description .

During "navigation", images are accessed through a number of operations which are logically associated to the concept GrOb (Graphical Object) and actually provided by the Multimedia Object Manager. Each time the user is positioned onto an instance of GrOb, these operations are made available. The main operations: display, select (selection of a portion of an image), hide (hiding of the image). In Fig. 4, the operation of "display" is performed on an instance of "Drawing", which is a specialization of the GrOb concept. The image in this case is an AutoCAD draft. In spite of the different formats, images are modelled uniformly as consisting of portions which can be selected by a common operation "select". In Fig. 5 it is shown the result of a "select" operation performed on the previous drawing. The GrOb describing that portion is displayed as a form and added to the iconic browser. The draft associated to the GrOb is finally displayed.



anything: B 19-12-93
code: d807-113
date: 19-12-1993
description: CANNOT POWER UP
name:
relative_to: control_panel01

Figure 3. Desktop screen : the iconic browser and the forms structure

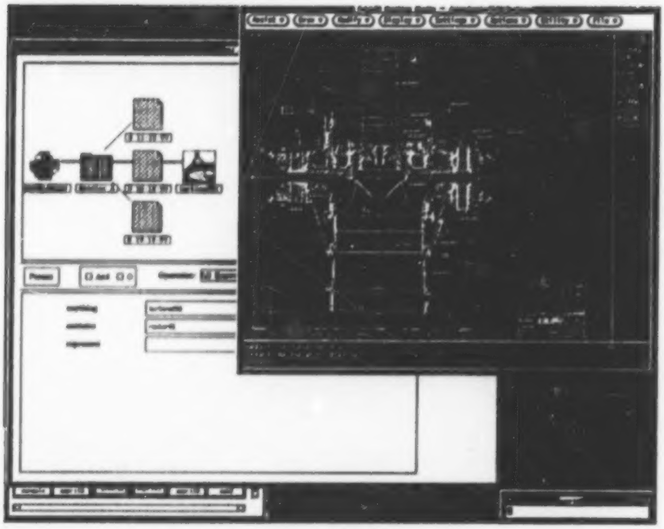


Figure 4. Navigation across images: display of an image

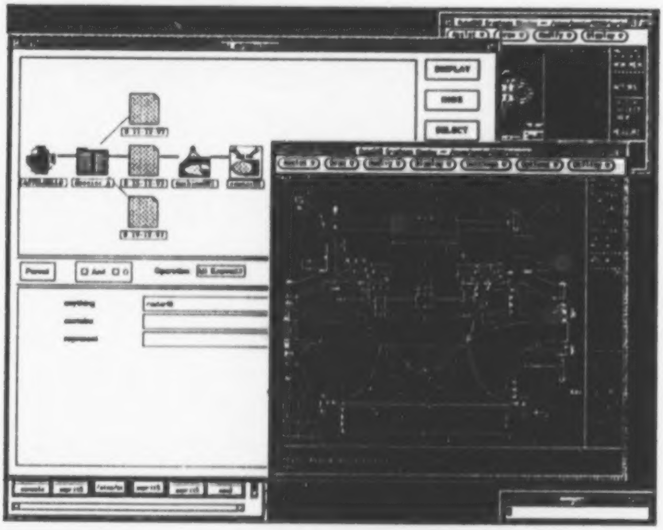

DISPLAY
HIDE
SELECT

Figure 5. Navigation across images: selection of an image portion and moving to the corresponding object

5. CONCLUSION

In the AIMS project it has been defined an intelligent information management system for the development of exploration-oriented applications, that is applications entailing indexing and free consultation of complex structured information bases. A description language is used for building the conceptual model of the information base which is visualized to the user through the IDA Desktop.

One of the goals of AIMS was to assess the effectiveness of the BACK system (more in general of description logics) for the development of real application. The issue is challenging and widely debated in the research community as witnessed by the increasing number of application oriented papers. Nevertheless very often we cannot find in these papers a clear description of which representation constructs are actually useful for doing what. That would be important also from a didactic point of view because of the expensiveness of training on description logics. Description systems are very facinating for their formal base but difficult to use in practice. The questions which typically arise are: when is it convenient to use a defined concept or role instead of a primitive one? What is the usefulness of the classification inference?

We do not believe that univocal answers are to be searched for. The experiences of use can be very different. Nevertheless it is helpful trying to classify the way these systems are applied. From the IDA experience it turns out that, in essence, we have used description logics as an advanced and operational semantic data model equipped with inferential capabilities. The inferential capabilities are exploited for performing: automatic and content based indexing of instances, part_of modelling, flexible navigation of instances. The constructs which are mostly used are: defined concepts and rules for automatic indexing of instances; defined roles for the intelligent navigation of instances and part_of modelling; constraints. Note that in contrast with many current applications relying on description logics, we do not make an extended use of concept classification. We consider concept classification more as a facility for the application developer - because the system takes charge of making the necessary checks on the consistency of the taxonomy - rather than an inference benefiting the user. A final remark is about some challenging but very desirable - from an application point of view - extensions of the system. The first is on constraint management. Especially it would be very important to express constraints on the knowleability of objects. The second requirement is on a more efficient management of persistency.

We conclude with some notes about the status of implementation: a prototype of IDA comprehensive of all the modules previously described have been developed and the application for technical documentation management has been implemented for demonstration purposes.

6. REFERENCES

Bertino, E., Damiani, M., et al., (1992), "Multimedia data handling in a knowledge representation system", Proc. Second Far-East Workshop on Future Database Systems, Kyoto (Japan).

Bertino, E., Damiani, M., et al., (1992), "A View Mechanism for a Knowledge Representation System, Proc. of the First International Conference on Information and Knowledge Management ", Baltimora (US).

Bertino, E., Damiani, M., et al., (1993), "An Advanced Information Management System", Proc. Workshop on Interoperability among Multidatabase System IEEE Research Issues in Data Engineering, Wien (Austria).

Brachman, R., Halper, F., (1992), "Knowledge Representation Support for Data Archeology", Proc. First International Conference on Information and Knowledge Management, Baltimora (US).

Brachman, R., Shmoltze J., (1985), " An overview of the KL-ONE knowledge representation system", Cognitive Science.

Damiani, M., Bottarelli, S., (1990) "A Terminological Approach to Business Domain Modelling", Proc. of the First International Conference on Database and Expert System Applications - DEXA 90, Wien (Austria).

Damiani M., (1993), " Representing the Part Of in Back: a Modelling Exercise", AIMS Final Deliverable, Milan (Italy).

Hoppe, T., Kindermann C., et al., (1993), "BACK V5 - Tutorial and Manual,KIT Report", Technical University Berlin, Berlin.

Ioannidis, Y., Linvy, M., (1992), "Conceptual Schemas: Multi-faceted Tools for Desktop Scientific Experiment Management", International Journal of Intelligent&Cooperative Information Systems.

Kim, V., Bertino E., (1989), "Composite Objects Revisited", Proceedings ACM Sigmod.

Menaglio, P., Spampinato L., et al., (1993), "A Language to Manage Database Information with a Treminological System, AIMS Final Deliverable, Milan (Italy).

Patel-Schneider, P., et al., (1990), "Term Subsumption languages in knowledge representation", The AI Magazine, 11(2).

Von Luck, K., Nebel B., et al., (1987), "The anatomy of the BACK system", KIT Report 41, Department of Computer Science, Technical University Berlin, Berlin.

THE DEVELOPMENT OF A DATABASE (FDB) FOR INFORMATION EXCHANGE AMONG CLINICAL ENGINEERS[*]

Aikaterini Krotopoulou[1], Georgia Panagopoulou[1], Spiros Sirmakessis[1], Paul Spirakis[1], Vassilis Tampakas[1], Dimitra Terpou[1], Athanasios Tsakalidis[1], Nicholas Pallikarakis[2]

[1]Department of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece
[2]Institute of Biomedical Technology, Ellinos Stratiotou 50A, 26441 Patras, Greece

Abstract

FINE DataBase (FDB) is a document database that keeps, in a structural way, information suitable for effective communication and co-operation among Clinical Engineers. The majority of this information is free texts with biomedical interest. An electronic magazine is also provided to the users, while additional data concerning medical devices, the user themselves and Clinical Engineers all around the Europe can be found in FDB. The goal of organising the above information is to assure the dissemination and the effective use of the gained experience as well as to strengthen the collaboration among Clinical Engineers. This paper gives a description of FDB concepts and design, which may serve as a framework for other specialised databases which deal with free texts. Furthermore, it gives an overview of the basic functions provided by the FINE DataBase.

1. INTRODUCTION

Nowadays a great amount of information relative to Clinical Engineers is produced by different organisations and institutes. Information concerning medical devices such as failures, malfunctions, solutions to technical problems, reports from European or other international organisations, quality control standards etc. is provided by many resources. Furthermore, announcements for new products, seminar calls for participation and bibliographic notes are distributed from different means. Every Clinical Engineer interested in these topics can be informed by reading relative magazines, conference proceedings or by communicating by means of every source of the above information.

As it can easily be seen, a lot of valuable time is spent in search of this kind of information. FINE DataBase (FDB) can reduce the time spent, because it is a tool that contains all this kind of information and supports the exchange of ideas in the Clinical Engineering area.

FINE DataBase is a tool that is being developed under the **B**iomedical **E**quipment **A**ssessment and **M**anagement (BEAM) AIM project. It is part of the **F**acility for

---

[*] This work was supported by AIM A2001 CEC Project (BEAM)

INformation Exchange (FINE) tool. FINE is a BEAM sub-system aiming to develop a prototype conferring tool to be used for information exchange in the area of Biomedical Technology. The basic system's principles is based on the need for communication's support among Clinical Engineers and other professionals. Its objective is to provide to the Clinical Engineering Community a central source of services, information and news over the existing Public Data Networks or Public Switched Telephone Networks. Therefore, Clinical Engineers even though they are physically isolated, they can form a distributed working environment.

This paper is an overview of the FINE DataBase. In section 2, there is a brief presentation of the information stored into FDB. Section 3 contains a description of the design and architecture of the database. The way a clinical engineer can access FINE DataBase is described in section 4. Section 5 contains a short description of the most important features of the system. The final section summarises our conclusions for the work done till now and gives in short our future plans.

2. AN OVERVIEW OF THE FINE DATABASE

Fine DataBase is a document database that keeps in a structural way free texts of Biomedical interest. According to the needs of Clinical Engineers, various kinds of free texts need to be supported. Consequently, texts have to be distinguished into categories and this has to be made in a way that guarantees the effective communication and co-operation among them.

According to a widely accepted consideration, free texts can be divided into *reports* and *announcements*. Reports, in their turn, can be distinguished into *official* and *general content* reports. Official reports are well constructed and approved documents provided by organisations or any authorised person. These reports are addressing issues related to accident alerts, quality control protocols and maintenance for any kind of medical devices included in FINE DataBase. Moreover EC Directives and International Standards can be found. On the other hand, general content reports are free texts provided by the FINE DataBase users, as a result of the interaction among them, reflecting their experience related to the use of Biomedical equipment. These reports can be relative to any kind of information that Clinical Engineers consider that is of interest to their community. General content reports can be distinguished in a way similar to the official ones.

As far as announcements concerns, it has to be pointed out that they are actually working as real life announcements, while they are provided in order to encourage the communication and co-operation among FINE DataBase users. Following the idea of effective division of texts into categories, seven kinds of announcements have been specified: announcements for bibliography, new products, meetings, seminars, expositions, job openings and collaborations. Any new book or product can be presented in the corresponding announcement while meetings, seminars or expositions are announced. Job openings and themes for collaboration can also be propound.

The existence of an *electronic magazine* was found to be essential for the promotion of information exchange, which is the basic goal of FINE system. Electronic magazine actually mimics the structure of "real" magazines. In other words, it is edited in issues, while each issue is formed by articles being free texts. Each issue can be distributed to every subscriber who can be any FINE DataBase user.

During the specification of FDB user requirements, the need of having additional information which is not structured as texts was detected. This information mainly refers to FINE DataBase users and Biomedical devices. As far as FDB users concerns, information like their names, addresses and other contact information, or more specific one, like their specialities, is kept.

3. THE FDB DESIGN

In order to achieve efficient storing and manipulation of FDB data -especially of free texts- special techniques have been considered. The main FDB design task was the achievement of a proper organisation of free texts. This has been firstly succeeded by the efficient use of basic data as indices for the documents. In other words, special basic data have been selected to represent the *basic keywords* of the texts, according to which the retrieval of the texts is performed. Basic keywords are relative to the content of the texts and have been carefully detected, while different basic keywords are defined for different text kinds. The number of basic keywords has such a value that both efficient representation of the content of the text is achieved and information redundancy is avoided. To have an idea about how the basic keywords are formally defined for every text kind, we give the representative example of the definition of the basic keywords of accident alert reports. They are indicated by the following entity set:

Define entity set accident_alert (text_id, model_id, failure, event_date, insertion_date)

The attribute "text_id" indicates the code of the report, while the rest attributes define the basic keywords which indicate the medical device ("model_id") on which the certain failure ("failure") took place on the defined date ("event date"). Finally, "insertion_date" indicates the date that the report was inserted in the FINE DataBase.

In a second step, in order to achieve efficient organisation of the free texts we gave to the user the ability of specifying some additional keywords that he assumes that indicate the basic concepts touched on the text. They are called *free keywords*. Furthermore, the subject of the text is kept. The formal definition of free keywords and subject of accident alert texts is indicated by the following expression:

Define entity set acal_text (text_id, (keywords), subject, file_name)*

The attribute "file_name" indicates the full path of the file containing the text. This is a special technique which was adopted in order to override the limitations of the traditional RDBMS's concerning free texts. More precisely, texts would not be stored as tuples of the FINE DataBase. Instead, only the filename of the file containing the text is saved. In this way we can handle documents of any size, containing any kind of information (images, graphics, pure text).

Additionally, information retrieval techniques, can be integrated with the FDB design for the efficient indexing and retrieval of free texts. The use of Vector Space Model, enriched with statistical methods for automatic indexing, and the Global Thesaurus, is expected to increase both the recall and the precision of the database. Especially for the Thesaurus we are going to support the system with a Global Thesaurus with automatic construction based on Document Clustering due to the connected components evaluation method.

The way we confront the structural organisation of free texts makes the representation of electronic magazine an easy task, since every issue is a collection of articles (free texts). The entity-set which indicates the organisation of electronic magazine is the following one:

Define entity set electronic_magazine (issue_id, issue_date, (subject, file_name))*

As far as non text-structured information concerns, no representation anomalies occur. A representative entity-set expression of this category of FDB information is given below:

Define entity set device_model (model_id, model_name, manufacturer, model_description)

The only point that has to be mentioned here, is that this information is closely related to the content of free texts, as it can be easily seen in the "accident_alert" entity set. All the entity sets and relationships that organise FDB information, are presented in the FDB entity-relationship diagram (Figure 1).

Another issue that has to be mentioned has to do with security aspects in the design of FDB. Due to the fact that texts can contain anything and users are allowed to insert any new text, the database should have mechanisms to protect itself from having garbage or irrelevant information. Every text is marked with the "id" of the user who inserted it. This is indicated by the following relationship:

Define relationship user_texts (user_id, text_id)

If something is wrong or irrelevant, using the above relationship, any available information about the author can be retrieved. Furthermore, any text can be retrieved by any user having this right, but it can be modified or be deleted only by the author. This effort is controlled by the same relationship.

Access rights to database functions are defined by the assignment of a user into one or more of four security groups. Different groups are granted with different access rights. Crucial operations like deletions are provided only to authorised users. Every user is assigned into one or more groups by inserting his unique "user_id" into the tuples of the "group" entity set.

4. ACCESS TO FINE DATABASE FUNCTIONS

FINE DataBase is actually a tool, specially designed for use under the FINE system. In this way, only FINE users can access the database. When a user wants to access the database, he has to enter into the FINE system, by means of a typical login procedure and then select the *"Access to FDB"* button for entering into the database.

The basic screen of the database, called "Main Window" of the FINE DataBase (Figure 2), is divided into two main areas.

The top area consists of five selections, representing the corresponding data categories existing in the FDB, while four buttons, describing the operations that can be executed, lie in the bottom area. In order to perform a FINE DataBase function, a user has to select one data category and then push one of the buttons for the chosen operation. A sequence of forms and menus appear, in order to guide the user to the exact definition of the function he desires. In this way, all FDB functions can be executed.

Figure 1. FDB Entity Relationship diagram

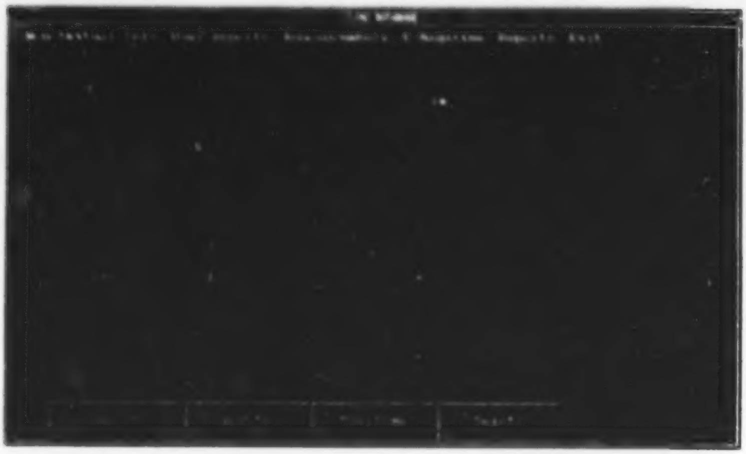


Figure 2: The Main Window of the FINE DataBase

An FDB user can execute the allowed functions, according to his rights. The database can recognise every user, after logging in, and grants the access rights according to the group or groups the user is a member of. Menu options and buttons, which are not available to a certain user, are locked (displayed in grey form), while the available ones are unlocked (displayed in white form).

Finally we must mention, that the graphical user interface has been designed in a way that actually guides the user by locking and unlocking the options, that can be selected at every step, following the BEAM common user interface guidelines.

5. OPERATING ON THE FINE DATABASE. A SHORT LOOK.

Fine DataBase contains information about biomedical data and gives to the users effective methods for accessing it. These methods mainly refer to the way we handle operations related to free texts. According to the consideration that texts are divided into reports and announcements, we illustrate the basic FDB operations.

Consider a FINE DataBase user who wants to perform an operation related to a FDB report. In order to perform this, he must choose the data category "*Reports*" from the Main Menu and define the kind of Report and the certain operation. In the case, for instance, that the selected operation is *Insertion*, a form prompting for the basic and free keywords, the subject of the report and the insertion date, is displayed. The user defines the values for these keywords actually by typing them, while embedded integrity constraints ensure that the most significant ones have not null values. Finally, the user can insert the new text using the FINE DataBase editor which provides all the basic editor functions.

In the case of *Retrieval* operation, a form prompting for the value of keywords according to which the retrieval will be performed, appears. These keywords are a subset of the basic ones, while they depend on the selected kind of Report. In all cases of reports,

BEST COPY AVAILABLE

the user can search for the desired one, by defining the insertion period of the report. This is a helpful process for someone who is periodically looking for new reports. Furthermore, the device which is related to the specific report can be defined as a search keyword. Other keywords can also be defined, depended on the kind of the report. For example, if the user wants to retrieve an Accident Alert Report, the form presented in Figure 3, is displayed.

Once the user has defined the keywords according to which the retrieval will be performed, a list containing the insertion date and the subject of each report satisfying the conditions, is displayed. Having this information, the user can verify which is the most interesting Report in order to read it.

Deletion and *update* operations are similar to the retrieve one but they are available only to the owner of the specific report (the user who has inserted the report), or to a person responsible for the proper function of database (e.g. DataBase Administrator). Operations related to *Announcements* are handled in a way similar to the one presented for the case of reports. Every user can insert an announcement in the DataBase informing the database community for common interest subjects. Also every user can retrieve any existing announcement either by specifying the keywords or the author of the announcement. Concerning the keywords, the user can also use partial matching in order to retrieve the proper set of documents.

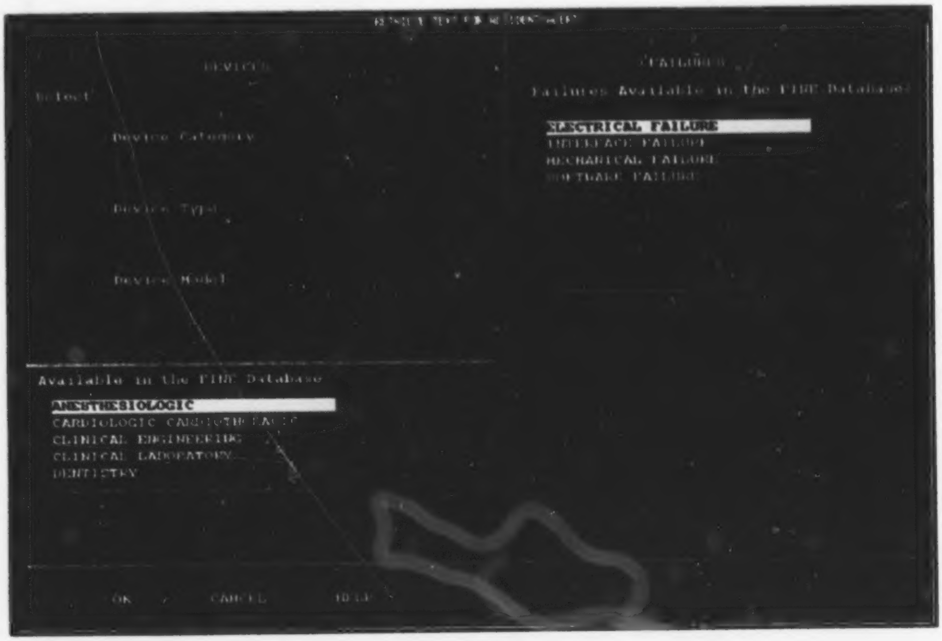

DEVICES
Select
Device Category
Device Type
Device Model
FAILURES
Failures Available in the FIRE Database
ELECTRICAL FAILURE
INTERFACE FAILURE
MECHANICAL FAILURE
SOFTWARE FAILURE
Available in the FIRE Database
ANESTHESIOLOGIC
CARDIOLOGIC CARDIOTHORACIC
CLINICAL ENGINEERING
CLINICAL LABORATORY
DENTISTRY
OK
CANCEL
HELP

Figure 3: An Accident Alert Report Form of Keywords

In the case of *Electronic Magazine* an Editorial Group is responsible for editing and broadcasting of the Electronic Magazine to the subscribers. Consequently, the operations of insertion, update and deletion, can be performed only by the members of the

EST COPY AVAILABLE

Editorial Group. Under the retrieval operation, two options are available: *Broadcasting* and *Reading*. Using the first option, the responsible member of the Editorial Group can distribute a new issue (that is ready) of the Electronic Magazine to the subscribers of that issue. With the second option, a FDB user can retrieve any issue of the magazine to which he is a subscriber. More precisely, a list containing the dates of the issues to which he is a subscriber, appears. The user can select one, and read any article of the issue.

Structured information (non-textual) is also kept, in FDB. This is basically information relative to devices, FDB users, clinical engineers etc. The operations of insertion, deletion and update of such data are performed mainly by authorised persons.

6. CONCLUSION AND FUTURE WORK

The purpose of the FINE DataBase is not only to make information available to users but to succeed that in a trustworthy manner. The implementation of the FINE DataBase fulfils a number of requirements imposed during the design phase like the need for efficiency, multi-tiered security, access rights, integrity and data independence.

Though it has been used a traditional DBMS (like the RDBMS of Oracle) for database operations concerning mainly text management and control, a surprising level of efficiency has been achieved. To our opinion this has been succeeded a) by the proper organisation of the documents and b) by the efficient use of the basic data as indices of the documents.

Our future work has mainly three directions:

- To implement a distributed version of the FINE DataBase. This extension will provide to the users a flexible tool for retrieving up-to-date information from any European region.
- To enrich the FINE DataBase with sophisticated information retrieval techniques. The use of these techniques combining with thesaurus and dictionaries will increase the recall and precision capabilities of the system.
- To adopt and use document standards (e.g. from ISO or ANSI) for the architecture, interchange and mark up of the documents.

7. REFERENCES

Amur R., Mohan A., Ramaswami M., TMS: A Free Form Text Management System, *Software Practice and Experience*, 1990, Vol. 20, No. 3, pp. 321-324.

Clifton C., The design of a document database, *ACM Conference on Document Processing Systems*, 1988, pp. 125-134.

Fuller M., Mackie E., Sacks-Davis R., Wilkinson R., Coherent Answers for a Large Structured Document Collection, in proceedings of *SIGIR 1993*, pp 204-213.

Jasinski P., Meinzer H., Sandblad B., An Approach to Integrate Collections of Images and Relational Databases, *Methods of Information in Medicine*, 1988, Vol. 27, No. 4, pp. 177-183.

Korth E., and Silberschatz A., *Database system concepts*, McGraw-Hill, University of Atlas at Austin, 1986.

Krotopoulou A., Lafazanis M., Panagopoulou G., Sirmakessis S., Spirakis P., Tambakas V., Terpou D., Tsakalidis A., Design of the S/W components of the FINE system and system configuration, Chapter 5, *Deliverable #13*, AIM/BEAM, FINE no A2001, December 1992.

Krotopoulou A., Lafazanis M., Panagopoulou G., Sirmakessis S., Spirakis P., Tambakas V., Terpou D., Tsakalidis A., FINE: The Pilot Phase Demonstrator, Chapter 5, *Deliverable #21*, AIM/BEAM, FINE no A2001, October 1993.

Krotopoulou A., Lafazanis M., Panagopoulou G., Sirmakessis S., Spirakis P., Tambakas V., Terpou D., Tsakalidis A., Vlahopoulos P., The Design of a Text Database for Biomedical Information (FINE DataBase), in the Proceedings of *1st International Conference on Medical Physics and Biomedical Engineering*, Nicosia, Cyprus, 1994.

Krotopoulou A., Panagopoulou G., Sirmakessis S., Spirakis P., Tambakas V., Terpou D., Tsakalidis A., Pallikarakis N., FINE DataBase: A Database for Information Exchange Among Clinical Engineers, in the Proceedings of *Twelfth International Congress on Medical Informatics, MIE 1994*, Lisbon.

Lafazanis M., Mamalis V., Spirakis P., Tambakas V., Tsakalidis A., FIRE: An efficient Information Retrieval System for Large Text Collections, *T.R. 94.01.5*, Computer Technology Institute, Patras, Greece, to appear in RIAO'94.

Salton G., Automatic text indexing using complex identifiers, *ACM Conference on Document Processing Systems*, 1988, pp. 135-144.

Stonebraker M., Stettner H., Lynn N., Kalash J., Guttman A., Document Processing in a Relational Database System, *The INGRES Papers*, 1983, Wiley.

THE DEVELOPMENT OF AN INFORMATION SYSTEM FOR HUMAN RESOURCE MANAGEMENT

Janez Jereb

University of Maribor, School of Organizational Sciences, 64000 Kranj, Prešernova 11, Slovenia

Abstract

The paper contains a description of the model of a human resources development information system. The model is based on the methodology of making individual development plans (career planning). A description of an appropriate relational database, created with Microsoft Relational Database Access, is discussed in brief.

1. INTRODUCTION

One of the basic presumptions in the concept and practical solution of the Human Resource Development Information System is its connection with the Organizational Information System. A further presumption is the creation of a database for employee's personal data and the creation of suitable modules or areas within the framework of the Human Resource Information System. We would like to define individual connections and areas in general and in detail those elements of HRIS, which directly support the module - Development and Education of Personnel (HRDIS).

2. CONNECTION BETWEEN THE ORGANIZATIONAL AND THE HUMAN RESOURCE DATABASE

The organizational database, is generally based on a detailed description of work positions. From the aspect of HRDIS it includes numerous, interesting data and information (see Fig. 1). Of special interest is the microorganizational aspect, which includes:

- jobs list or work positions list
- general or internal nomenclature of professions
- creation of positions
- standardization of positions
- setting goals and tasks for positions.

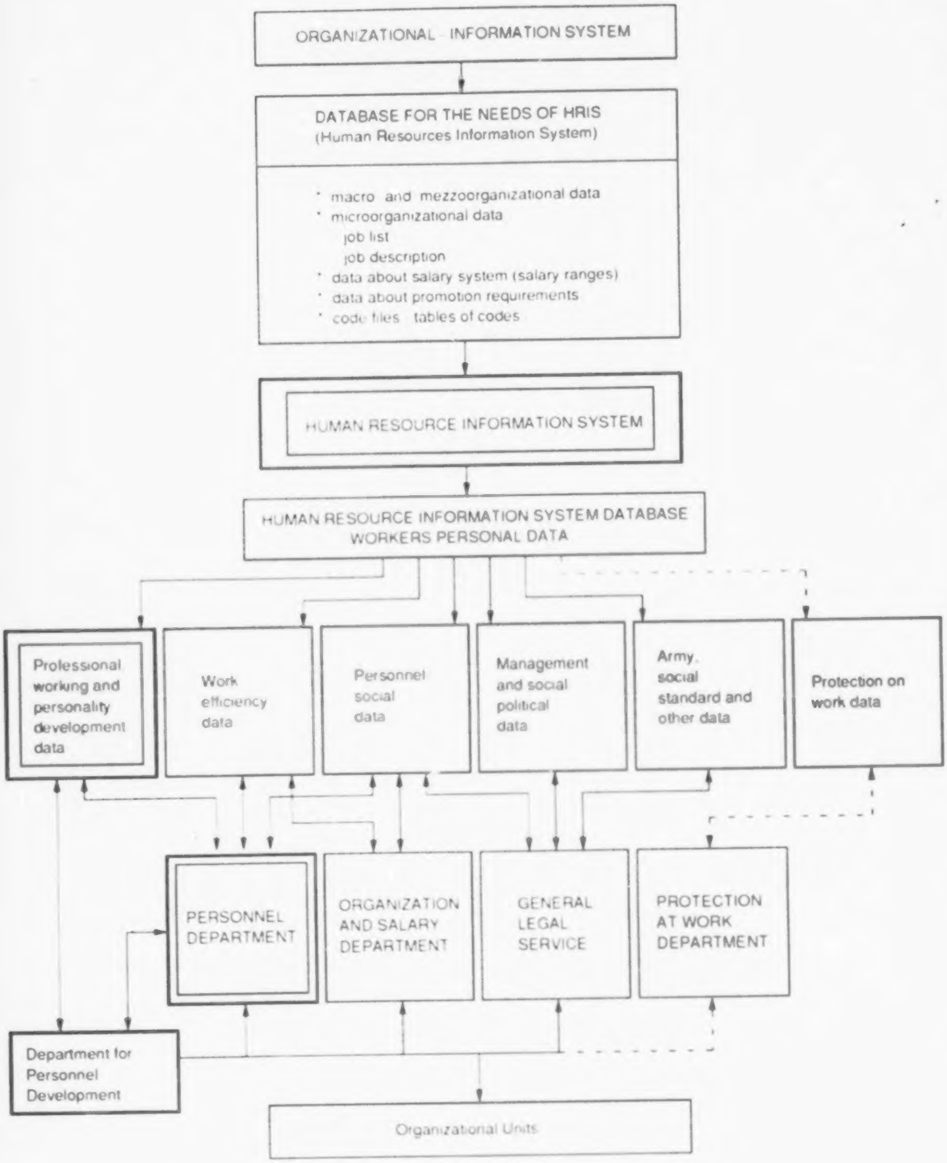

ORGANIZATIONAL - INFORMATION SYSTEM
DATABASE FOR THE NEEDS OF HRIS
(Human Resources Information System)
* macro and mezzoorganizational data
* microorganizational data
job list
job description
* data about salary system (salary ranges)
* data about promotion requirements
* code files - tables of codes
HUMAN RESOURCE INFORMATION SYSTEM
HUMAN RESOURCE INFORMATION SYSTEM DATABASE
WORKERS PERSONAL DATA
Professional working and personality development data
Work efficiency data
Personnel social data
Management and social political data
Army, social standard and other data
Protection on work data
PERSONNEL DEPARTMENT
ORGANIZATION AND SALARY DEPARTMENT
GENERAL LEGAL SERVICE
PROTECTION AT WORK DEPARTMENT
Department for Personnel Development
Organizational Units

Figure 1: Concept of the Human Resource Information System

BEST COPY AVAILABLE

For the purpose of accompanying and developing human resource, those data are important, which are part of a detailed description of an individual position or part of individual tasks and assignments. In rough these data can be divided into the data about work and the data about the employee (human resource data: data about the conditions for stipulating employment and the data about promotion requirements).

HRDIS presupposes above all the following data:

a) Data about work, tasks and assignments:

- kind of work (management, performance, cooperation, control etc.),
- work area or location,
- objects of work, work equipment, protection equipment/facilities,
- microworking environment,
- tariff group,
- work/task assessment - pay class,
- expected work results etc.

b) Employee's Data - Work Requirements

- professional qualification (level of education, profession),
- work experience,
- functional and other additional knowledge and skills,
- other requirements (trial work, qualification test, suitability for work etc.),
- responsibility, social working conditions,
- special psychophysical abilities,
- actual work results etc.

The carrier of organizational data is usually the position, which is defined by a suitable code. Individual work positions/tasks and assignments are coded as well. Individual jobs or tasks and assignments (with all the belonging data) are then connected with the help of the employee's register no. in the HRIS database with individual employee (employee's personal data). Here either uniform register no. of citizen or internal code can be applied. In short, the organizational database and the human resource database have to be suitably connected.

3. THE HRDIS DATABASE

The basic database (employee's personal data), can be organized in different ways. It is wise to create at least three main data groups: Basic Personal Data, Data about Residence and Employment Data. From the aspect of human resource developmnent the concept of HRIS should ensure foremost the possibility of data acquisition about:

- employees
- probationers
- scholarship holders
- candidates, applying for a job

Specific areas of personnel activities need specific data, therefore the main data groups and belonging subdata groups must be defined to provide necessary support for all areas of human resource development and training activities.

The structure or the logical model of the relational HRDIS database is presented on the next page. In the graphic presentation of the logical model one can see the basic tables and their logical connection (see Fig. 2).

The graphic model does not show the relations for the following tables: Programs, Files and Backing Tables. These are relatively independent tables, which act as outside keys in individual tables. If all the files and backing tables with their relations were drawn in the scheme, an overview would be lost, therefore their graphic presentation has been omitted.

3.1. Table and Relations Review

The basic structure of each individual table consists of: Field Name, Data Type and Description. For the individual field within a table various Field Properties can be defined, such as: Field Size, Format, Caption, Default Value, Validation Rule, Validation Text and Index. All these data can be seen in the Table Design View.

From the base structure it can be seen that one key covers the following three entities: Employee, Probationer and Scholarship Holder (individual entities are defined by a special status). Thus the majority of the tables is the same for all the three mentioned entities, their only difference being in Probationer and Scholarship Holder entities, which do not show relations with all the tables and which contain certain specific tables that are not related to the Employee entity.

Data for the Job - Work Position entity can be found in the specific tables like: Job - Tasks description, Job - Special conditions, Job - Skills, Job - Training and other Job tables.

The Programs table contains various data about internal and external training programmes. Basically this table has been created as a dynamic file which can be constantly supplemented.

Tables of Codes in the HRD database are divided into three basic groups: human resource (CTh), organizational (CTo) and general (CT). Some of them have already been created and entered into the databes, others have to be created and defined by the user.

3.2. Form Review and Description

For work with data within individual tables and relations the database includes various forms (displays). Window structure is the same for all forms. It consists of:

- window head,
- pull-down window menus,
- window names with command icons (keys),
- form head with command icons (keys),
- middle part with data fields,
- navigational keys,
- information about the form and individual form fields.

Because of their number the forms in the HRD database are divided into more levels according to width and depth.

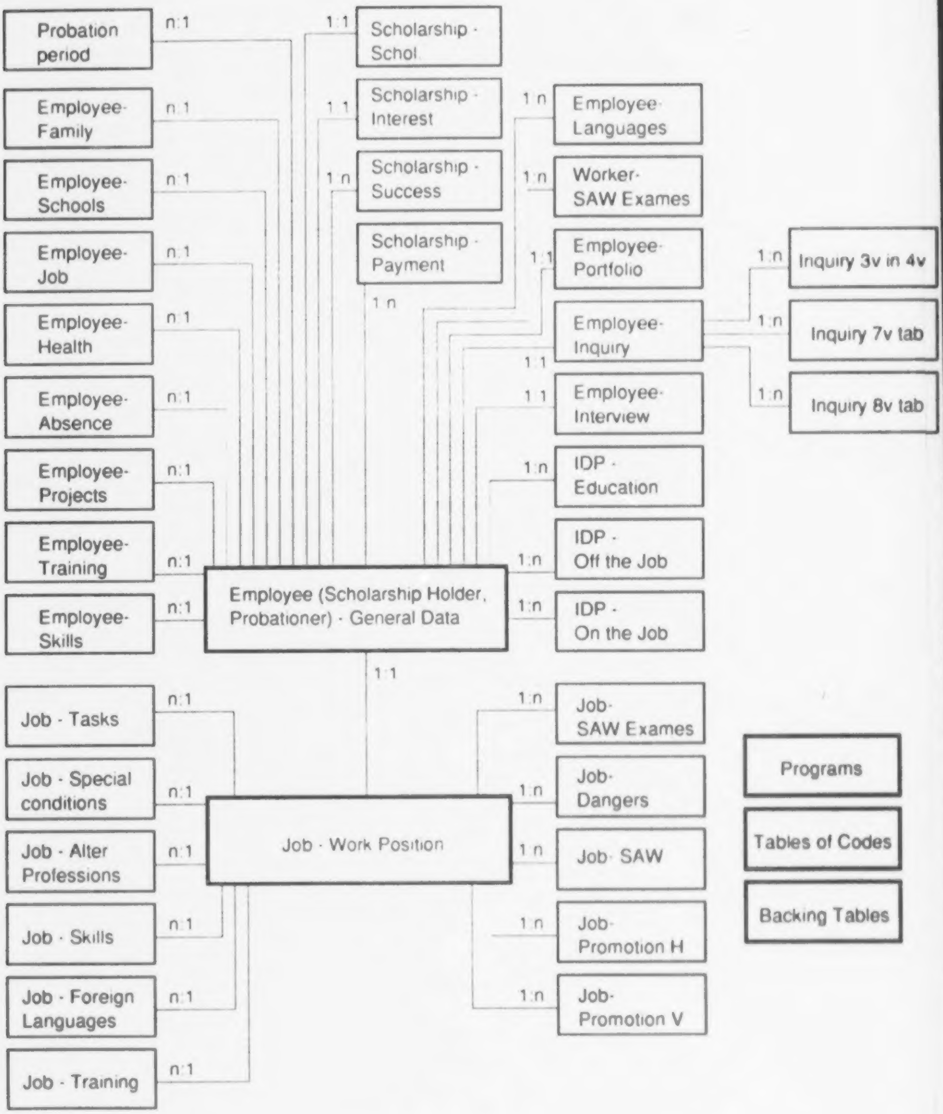

Probation period
n:1
Employee-Family
n:1
Employee-Schools
n:1
Employee-Job
n:1
Employee-Health
n:1
Employee-Absence
n:1
Employee-Projects
n:1
Employee-Training
n:1
Employee-Skills
n:1
1:1
Scholarship - Schol
1:1
Scholarship - Interest
1:n
Scholarship - Success
Scholarship - Payment
1:n
1:n
Employee-Languages
1:n
Worker-SAW Exames
1:1
Employee-Portfolio
Employee-Inquiry
1:1
1:n
Inquiry 3v in 4v
1:n
Inquiry 7v tab
1:n
Inquiry 8v tab
1:1
Employee-Interview
1:n
IDP - Education
1:n
IDP - Off the Job
1:n
IDP - On the Job
Employee (Scholarship Holder, Probationer) - General Data
1:1
Job - Tasks
n:1
Job - Special conditions
n:1
Job - Alter Professions
n:1
Job - Skills
n:1
Job - Foreign Languages
n:1
Job - Training
n:1
Job - Work Position
1:n
Job-SAW Exames
1:n
Job-Dangers
1:n
Job-SAW
1:n
Job-Promotion H
1:n
Job-Promotion V
Programs
Tables of Codes
Backing Tables

Figure 2: HRDIS Database Structure

BEST COPY AVAILABLE

3.3. Query Review and Description

With queries the user searches for different data which are stored in individual database tables. In practice Select Queries, are mainly used. With their help data can be checked, analyzed and changed. Here either data from one table only can be used or they can be combined from two or more tables.

Since the needs for queries are numerous and varied, it is advisable that they are created by the user according to his specific needs. Standard Queries, frequently used in the HRD database are created in advance and can simply be chosen from the appropriate query menu.

When we want to order selected records in a certain sequence, we create Sorting Queries. If we want to choose some records from a table following a certain criteria, we use Filter Queries. Frequently a combination of both (i.e. Filter/Sort Queries), can be used.

After creating and storing specific queries, the user can create a new query which helps him search for data in the first query. This is creation of Queries on the basis of queries.

3.4. Reports and Report Description

Since the needs for reports are numerous and varied, each user is advised to create them for his specific needs as he goes along. Standard Reports, frequently used within the HRD database, have been created beforehand. They can be used by simply choosing them from the corresponding report menu.

3.5. Macro Review

The basic structure of each macro command consists of: Macro Name, Condition, Action and Comment. For individual Actions different Action Arguments can be defined, for ex.: in OpenForm: Form Name, View, Filter Name, Where Condition, Data Mode and Window Mode. All the given data can be found in the Macro Design View, therefore they will not be discussed further.

4. SOME CONCLUSIONS

To design HRDIS it is usually necessary to form a suitable project team. The basic tasks of such a team are the following:

- creating personal human resource databases for all employees (as well as other groups, for example, probationers, scholarship holders, candidates);
- ensuring connection between personal human resource databases and other databases, especially the organizational database;
- defining basic areas of personnel activities as system users;
- defining information and processing needs for individual users (according to areas) as well as for other (outside) users.

For the homebase of human resource data as well as for individual human resource areas, it is necessary :

- to define input data specific for each area.
- to create input documentation for preparing and entering the data and to define the way of aquisition;
- to design files and instructions for filling in the documentation and for entering the data;
- to specify input documentation;
- to design input data forms;
- to plan, prepare and define kind of processing;
- to work out suitable programs;
- to design records, tables and forms for output information;
- to denote ways of output documentation;
- to prepare expert systems which serve as support for certain specific processing.

When creating HRDIS in a concrete organization the information (data) needs and specific processing of individual areas within human resource development and educational activities are first to be defined in a suitable preparatory study. To realize this requirement opinions and demands of individual users (especially individual departments and activities within the personnel departmentt) have to be collected, concerning data and processing they need or expect from HRDIS. For this purpose a suitable inquiry is prepared, which is to be answered by persons who are responsible for specific tasks and assignments within the personnel, legal, educational as well as organizational and renumeration departments. If necessary, the SAW (Safety at Work) department can be included as well. On the basis of data, opinions and proposals collected from individual users a final organizational concept of HRDIS can be created, which will meet the needs and expectations of individual concrete users.

An efficient HRD system must be supported by appropriate information system. We have described such an system created with MS Access 1.1. We hope we have described our system clearly enough, so that interested reader can built himself a picture about what our information system offer, made possible and which tasks in whole HRD system can support.

For future MS Access Programer we want to express some experiences to which we became during our work with MS Access 1.1:

- In general we cannot say which data type is more suitable for the key fields, but in Access seems to be an advantage if we use strings for key attributes. On one hand is a number alone offten not clear enough and on the other hand we can search those types of key attributes with wildcard. Namely, the wildcard character cannot be used with integer, long integer, or real fields.

- For many Microsoft programs we don't neccessary need a manual to understand them. But to work with Access without manual is, however, a great waste of time. It is much better to invest some time to study some global concepts first.

- By using subfiles in MS Access we must acount some restrictions. Subfiles can consume a lot of memory, and we can access them only trough the parent file (which means we cannot perform a search on a subfile). The best practice is to create subfiles only for information that demands multiple entries.

- It is a good convention to use the same layouts for entering and modifying records. The consistents control performed by data entry must have the same value for new and for old records, so this statement is clear.

- If a field in an output-layout would be displayd it is not allowed to change this field in any procedure. If we don't acount this a system break down can occur. To display such a field we must save it before as a variable.

- MS Access offer confortable standard form, query and report windows. These windows are easy to create with adequate wizards. A user can easyly costumize them for his special needs.

- MS Access is a powerful tool which permit realisation of complex information systems. We can characterise it as an confortable prototyping-tool for creating of efficient, specifically and user oriented applications.

5. REFERENCES

Armstrong, M., (1993), A Handbook of Personnel Management Practice, Kogan Page, London.
Davis, S., (1993), Teach Yourself Access, MIS Press, New York.
Hilb, M., (1984), Diagnose-Instrumente zur Personalentwicklung, Verlag P.H. Bern, Stuttgart.
Nadler, L., Nadler, Z., (1989), Developing Human Resources, Jossey Bass, London.
Microsoft Access User's Guide, Microsoft Corporation 1992
Microsoft Access Language Reference, (1992), Microsoft Corporation.
Perry, G., Access Programming By Example, (1993), Que, Carmel
Rickhof, H.Ch., (1993); Strategien der Personalentwicklung, Gabler Praxis, Wiesbaden.
Shammas, N.C., Microsoft Access Programming, Windcrest/McGraw-Hill.
Schein, E.H., (1985), Career anchors, University associates, San Diego.
Singer, M.G., (1990), Human Resource Management, PWS-Kent Publishing Company, Boston.
Zehnder, C., (1987), Informationssysteme und Datenbanken, Verlag fuer Fachvereine, Zuerich.

STRATEGIC INFORMATION SYSTEMS : A NEW PERSPECTIVE

James G. Howell

University of Paisley, Information Systems, High Street,
Paisley PA1 2BE, Scotland.

Abstract

The domain of strategic information systems (SIS) is broad and diverse.
The concern of SIS is the provision of information systems to support the organisation or business growth beyond short term considerations. The highest level purpose of information systems is to enable sound management decision making and consequent action to allow the business to continue to prosper.

Strategic Information Systems (SIS) does not refer to a type of information system but rather reflects a perspective on information systems, (Senn, 1992)

> "Information systems have strategic value, depending on how they are applied. The use of IT, not the system type, is the distinguishing factor".

During the 1970s, 80s and 90s significant research and literature has mainly related to large businesses.

Most of the existing research material relates to large organisations and corporations (for example, American Airlines' "SABRE", Minitel, BP, Ford,.....). There appears to be insignificant research in strategic IS involving smaller businesses. In this paper a smaller business will be described as a Small to Medium-size Enterprise (SME), a term which will be further defined in this paper. It is important that research includes these SMEs because, for many countries, the majority of the national wealth comes from these business organisations (Financial Times, 1993).

The purpose of this paper is to set out this evolution and to begin to question the applicability of this research to "smaller" businesses.

1. INTRODUCTION

Strategic management is concerned with taking an overall view of an organisation's situation and the business options available. Taking a strategic perspective requires taking account of full information on which to base ensuing decisions.
Strategic Management is (Bowman and Asch, 1989),

> "......the process of making and implementing strategic decisions... (it) is about the process of strategic change. (It is) the match an organisation makes between it's own resources and the threats or risks and opportunities created by the external environment in which it operates. So strategy can be seen as a key link between what the organisation wants to achieve - it's objectives - and the policies adopted to guide it's activities"

In the past, ready access to full information was not often possible due to application development type and nature; a type which was derived from the DP and IT eras of the 70s and 80s. These eras left behind a legacy of complex, proprietary and expensive IT environments which focused on data processing (transactional systems, databases,....) which continued to be based on formal structures of the organisation (Keen P G, 1991).

In strategic management, the effects of strategic decisions are, characteristically, felt throughout the business organisation and potentially determine the future health of the business. The advent of strategic IS has encouraged strategic planners to recognise the management of this business information, on which such decisions are made. Taking a strategic perspective, this means all the business information.

Associated information technology has developed at an ever increasing pace. It has,

- become more robust and reliable,
- become less expensive and significantly more powerful,
- become more open through improvements in communications (partly through standardisation),
- been distributed more freely throughout business organisations,
- become much more accessible and available,

and become easier for users to employ (Windows, spreadsheets, LANs, electronic mail, etc.). "Whether we like it or not, IT is becoming pervasive" [Earl M J, 1989, p13].

As information systems and associated information technology spread throughout businesses, the strategic planning and management of these resources becomes, necessarily, of significant importance.

2. STRATEGIC PLANNING.

Strategic planning involves planning for the business as a whole and in this paper is considered to be synonymous with corporate planning.

The corporate planning concept evolved from the 1960s and was refined to an implicit model in the 70s (Andrews, 1971) and further developed into the well recognised model of Argenti (Argenti, 1980), see Figure 1.

Figure 1.Argentis Model of The Corporate Planning Process

This model of Strategic Management is viewed as a process composed of three sub-processes (Johnson and Scholes, 1988, pp. 9-10), strategic analysis, strategic choice and strategic implementation. The model was subject to increasing criticism as being too mechanistic, laborious and over simplified (Porter, 1987; Earl M J, 1989a; Mintzberg, 1990; Quinn J B, 1989). Moreover, the application of this approach to strategic planning was not recognised as engendering strategic thinking .

The model suggests a prescriptive, deliberate, linear and detailed approach to corporate planning in an environment which was constantly subject to the ebb and flow of business events and business decisions .

Increasingly the planning process became seen to be irrelevant (Porter M E, 1987),

> "Line managers tolerated planning, but increasingly dismissed it as an irrelevant ritual".

3. STRATEGIC THINKING

Strategic planning was not recognised as a panacea for success. As Japanese companies became more successful, leaving American counterparts in their wake, the focus of strategic planning shifted to the perceived basis or important determinants of the Japanese companies' successes (Porter M E, 1987).

The Japanese companies' focus on quality, productivity and teamwork put the analytical basis of strategic planning into question. Although the need for good planning continues to be recognised the methods and techniques of strategic planning were reconsidered.

It became necessary to incorporate quality planning, the strategic recognition of company culture and workings, and the ability to engender entrepreneurialship into strategic planning. Engendering strategic thinking throughout an organisation continuously and linking strategic thinking (and planning) right through to implementation became better recognised as of vital importance. Strategy making is a continuous process (Porter,1987).

> "To be effective strategic planning must use a proper process, because strategy cannot be separated from implementation. Strategic thinking cannot occur only once a year, according to a rigid routine. It should inform a company's daily actions. Moreover, the information necessary for good strategic thinking is equally vital to running a business - designing marketing material, setting prices and delivering schedules, etc. Strategic planning must therefore become the job of line managers, not of head office staff ".

4. STRATEGIC MANAGEMENT

Strategic management is a continuous process involving change. Decisions made today affect the future, as past decisions affect the present. As the effect of decisions is recognised and quantified so the next generation of decisions should be better.

Strategy formation can be considered as an iterative and adaptive process [Cyert R M and March J G, 1963]; the product of a stream of decisions made by different individuals through time. This may, then, be described as "deliberately emergent" [Mintzberg, 1978].

In the past, strategic management was often described as a deliberate process; a process of analysis leading to a choice of decisions which were deliberately implemented. During the 1980s this description was increasingly challenged as being often inappropriate. This view did not recognise that many decisions would be better described as emergent. Strategy can then be viewed as a combination of deliberate and emergent decisions [Mintzberg and Waters, 1985].

A similar perspective to that of emergent decisions is an approach described as "Logical Incrementalism". From a study of the strategic change processes in 10 major companies (Quinn J B, 1980) concluded that the process of formal planning is simply one component in a continuous stream of events that serve to determine corporate strategy. The conclusions of this research, in brief, suggest,

- effective strategies tend to be emergent
- decisions tend to be undertaken incrementally
- the strategy-making process involves staff at many levels

The approach or perspective of Logical Incrementalism contrasts sharply with the "rational" perspective of Corporate Planning and does not, in particular, require the identification of objectives before undertaking strategy formulation.

5. STRATEGIC MANAGEMENT AND INFORMATION SYSTEMS

In this paper the concept of information systems is considered to be dependent on a chosen definition of data and information, (Stowell F A and West D, 1994).

> "Information is data to which meaning has been attributed in a particular context and an information system, in principle, serves the decision making process to enable consequent actions to take place".

Information systems exist in most business organisations at all levels. How the strategist views strategy making is dependent on his/ her view of an organisation. However, regardless of the view that is taken of the nature and form of an organisation, strategic decisions are taken during the course of business life. Since there is no unique rational basis for judging what is good or bad for a business, the strategist must be content with accessing the required information to support decision making in line with the business strategy and policies.

Information management is then, again, a key issue. It has been suggested that the relationship between strategic management and the management of IS is ambiguous with a bias towards systems and technology rather than information management (Sutherland E, 1992)

This may have resulted partly from the haphazard growth of information systems and associated technology within organisation, (Peppard J, 1992).

> "For many organisations IS/IT investment has tended to be piecemeal and haphazard. Coordination and integration of systems is very often not a consideration. This has been exarcebated in many organisations with the advent of cheap micros and local area network (LAN) software. Aligning IS/IT with existing business strategy requires the development of both an IS strategy (what is required in relation to information systems), and an IT strategy (how it is going to be delivered). The key is that business strategy drives IS/IT strategy.
> However, beyond using IS/IT to support existing business strategy, firms have the opportunity in using IT, proactively, to create new opportunities for the business".

In developing information systems to complement strategic thinking the strategist should recognise the nature of information, (Galliers, 1993a).

> "Because of the nature of information it follows that a reasonable approach to the development of a flexible information architecture must take into account the context in which necessary actions and decisions can take place, and the manner in which information is to be inferred form the data provided".

IS managers should be involved continuously in strategic IS development within the business working.

As information systems and associated technology become more pervasive throughout businesses and organisations, today's modern manager is engaged in making more decisions of a strategic consequence. Decisions which may have previously been considered as being of a tactical nature can have major strategic consequences. For example, (Earl M J ,1989),

> "Decisions made about hardware, software, data and systems can have major policy repercussions. For example, it is reported that two recent building society mergers in the UK fell through partly because the computer systems were incompatible".

Studies of the strategic process indicate that strategy is no longer the province of top management (Wooldridge, B and Floyd, S W, 1990; Ansoff, 1965; Andrews, 1971; Schendel and Hofer, 1979). Strategy has, for example, been described as the product of autonomous behaviour initiated outside of top management. Some studies have suggested the crucial role of middle-level managers in the strategic "process".

This underlying model of an organisation as a hierarchical management structure has been discussed widely (Checkland 1981, Hirschheim 1987, Jackson 1992) and often considered to be inappropriate to today's dynamic notion of an organisation as an open system with a purpose. Later in this paper we will return to the consideration of business organisations having a unitary purpose, mission or objectives.

In this paper we will adopt a view of an organisation as an *open system* (Vickers, 1983) in a continuous state of change. This open system is formed of cooperative human activities harnessed to achieve the explicit or implicit goals of the business.

6. ESTABLISHING ORGANISATIONAL OBJECTIVES

The definition of strategic management employed throughout this paper indicates that strategy can be seen (Bowman and Asch, 1989),

> "as a key link between what the organisation wants to achieve - it's objectives - and the policies adopted to guide it's activities".

Identifying and agreeing goals and objectives at different levels in an organisation can be a difficult task.There are many alternative views of "objectives" and "goals".
In this paper goals are considered to be those reasonably stable statements of organisational purpose which are normally characterised by some vagueness.

This term (goal) is similar to that of the mission or mission statement which is generally the organisation's highest level statement of unitary purpose.

Objectives are intended to be clear, measurable statements which are recognisably achievable. A typical hierarchical relationship exists between strategy and objectives (Ansoff, 1984).

> "Elements of strategy at a higher managerial level become objectives at a lower one".

Different employees are likely to have different goals (Capper, 1988).

> "Each person views the world in his or her own way and as such needs information that reflects these views. You have probably experienced the type of conflict that occurs between Operations,and Sales and Accounting, or between tactical and operational levels of management. Each wants information to support their own goals in this conflict, which is a reflection of each individual's own view of the world, the organization and sub-organizations".

Strategy requires to integrate these views. (Quinn et al, 1988).

"A strategy is the pattern or plan that integrates an organisation's major goals, policies and action sequences into a cohesive whole".

The assumed view of businesses and organisations consistently pursuing objectives which support their mission and goals consistently has been questioned as being over-simplistic.

Managers often face multiple and often conflicting goals, not a single unambiguous goal. In the development of Information Systems the successful project manager often locates the project within a Quality/ Cost/ Time triangle, illustrated in Figure 2.

Figure 2. Quality/Cost/Time Triangle

Organisations generally have multiple stakeholders ; employees, managers, shareholders, suppliers, customers and others.

Today's manager requires to be aware of meeting the needs of all of these parties and seeks to set objectives which reflect a consensus view.

Manager's decision-making is characterised by trade-offs, optimisation and profit-making.

7. STRATEGIC MANAGEMENT, INFORMATION SYSTEMS AND THE SME

This paper is seeking to consider the relevance of existing research on strategic information systems to the Small to Medium-size Enterprise (SME). Firstly we should seek to establish the criteria, if possible, of identifying an SME.

In the past the small firm has been defined as (Bolton, 1971) one operated as an independent business, managed directly by its owners and possessing a relatively small market share. Further sectoral, workforce and turnover definitions were specified. The term "small business" does not reflect a perspective since few, if any, businesses consider themselves to be "small" (Simpson R, 1993) in outlook.

More recent classifications of the small business include "a business of ten or less employees" (PC Business Software (UK), 1992), 25-500 employees (European Community award regulations).

It is now suggested that an SME may be considered to hold 200 or less employees and that **a count of key decision makers** will be an important measure (The Scottish Office, 1993; Henderson D, 1992).

Current SIS research requires a focus on the information needs of the business activities that support the business strategy since IT development should support the business strategy (O'Connor A D, 1993; Galliers R D, 1985). This is relevant to all businesses regardless of size or sector. This may be done by modelling the key business activities that determine the health of the business for some time ahead. Intrinsic to this approach is the recognition that strategic information systems are not discrete but permeate business organisations, perhaps haphazardly. Strategic IS is not monopolised by upper echelons of management but are constructed at different levels by decision-makers in the business. As suggested by Galliers (Galliers 1987, p6).

> "We should move the notion of organizational activity away from the hierarchical focus adopted by Anthony (1965) and Keen and Scott Morton (1978), already cited, to a managerial level-free typology".

This approach is more readily applicable to researching strategic information systems and the SME.

A jump requires to be made to the identification of information needs,
this is not easy (Earl, 1989).

> "The jump to determining information system needs seems to be more difficult".

It is important to adopt a perspective underlying the information needs identification exercise. An example, similar to that of Galliers (Galliers, 1987) of an underlying perspective suggests a,

- top-down
- infological
- activity or decision oriented (as opposed to data oriented)
- user/ manager oriented (as opposed to data processing specialist oriented)

approach. The next step of this approach or methodology is to model the business activity information needs. It is not within the scope of this paper to explore the possible approaches to doing this.

As the evolution from strategic planning to strategic thinking moves into the 90s the notion of achieving sustainable competitive advantage has evolved to include business relationships beyond competitive advantage (Galliers, 1993b). Collaborative and complementary business arrangements are not uncommon, and can provide a route to business success.

8. SUSTAINING COMPETITIVENESS AND BEYOND

During the 1980s the "crie de couer" of businesses was to strive to be more competitive in the **changing environment**, both external and internal.
Achieving sustainable competitive advantage is not dependent on the technology itself but on the *manner* in which the technology is deployed. (Galliers, 1993b; Senn, 1992)

The 1980s have been characterised by business restructuring. Businesses have reduced production costs, reduced the number of employees, changed organisational structures and extinguished functional areas, offered redundancies and early retirements, decentralised and created geographical business units, spread fixed costs across business units, flattened the organisational structure............................ in an attempt to become "*lean and mean*" to compete in today's environment.

All of these measures have not necessarily resulted in success. Restructuring is not a strategy in itself.

The IS development process incorporates business restructuring (Galliers, 1992)

9. STRATEGY- MAKING CONCERNS

Particular strategy-making concerns have been recognised in the literature. These include,

- the small percentage of top companies that have formal strategies (Warr, 1991)
- strategy studies are recognised as being expensive in both management time and in consultancy costs. A subsidiary concern is that the time taken for such studies is often too long, and hence devalues the final product. (Warr, 1991; Galliers, 1993a; Lederer and Sethi, 1988)

For such reasons, and others, it is widely held that most IS strategies are created and simply placed on the bookshelf. (Warr, 1991)

10. CONCLUSIONS

The evolution of strategic information systems follows the evolution from the DP to IT eras. In the information systems era of the 90s, a more holistic approach suggests that the research base of strategic IS is extended to include business organisations of all types and sizes.

It is important that the Small to Medium size Enterprise is included in this research.

11. REFERENCES

Andrews K, (1971), "The Concept of Strategy", Dow Jones, Homewood, IL.

Ansoff H I, (1965), "Corporate Strategy: An Analysis Approach to Business Policy for Growth and Expansion", McGraw-Hill, New York.

Ansoff, H I, (1984), "Implanting Strategic Management", Prentice-Hall, London.

Argenti J, (1980), "Practical Corporate Planning", Allen and Unwin, London.

Bowman C and Asch D, (1989), (eds),"Readings in Strategic Management", MacMillan, London.

Checkland P B (1981), "Systems Thinking, Systems Practice", Wiley, Chichester.

Cyert, R M and March J G, (1963), "A Behavioural Theory of the Firm", Prentice-Hall, Englewood-Cliffs, NJ.

Earl M J, (1989), "Management Strategies for Information Technology", Prentice-Hall, Cambridge.

Financial Times, (1993), "Businesses Complain of Bank Pains", 27 Sept. 1993.

Galliers R D, (1992), Soft Systems, Scenarios and the Planning and Development of Information Systems", Systemist, Vol. 14, No. 3, Aug '92.

Galliers R D, (1993a), "Towards a Flexible Information Architecture : Integrating Business Strategies, Information System Strategies and Business Process Redesign" in JIS, Vol. 3, pp. 199-213.

Galliers R D, (1993b), "IT Strategies: Beyond Competitive Advantage".

Galliers R D, (1985), "An Approach to Information Needs Analysis" in Human Computer Interaction - INTERACT 84, (Ed. Shackel), North Holland.

Galliers R D, (1987), "Information Analysis : Selected Readings", Addison-Wesley, Wokingham, England.

Keen P G, (1991), "Redesigning the Organisation Through Information Technology", Planning Review, May/June.

Henderson D, (1992), *for* The Scottish Office, Small Firms in Scotland report.

Hirschheim R A and Boland R J (eds.), (1987), "Critical Issues in Information Systems Research", Wiley, Chichester, UK.

Jackson M C, (1992), "Systems Methodology for the Management Sciences", Plenum, New York.

Johnson G and Scholes K, (1988), "Exploring Corporate Strategy", Prentice-Hall, Hemel Hempstead.

Lederer A L and Sethi V, (1988), "The Implementation of Strategic Information Systems Planning Methodologies", MIS Quarterly, 12, pp. 445-461.

Mintzberg H, (1990), "The Design School :Reconsidering the Basic Premises of Strategic Management", Strategic Management Journal, Vol. 11, pp. 171-195.

Mintzberg H and Waters J A, (1985), "Of Strategies, Deliberate and Emergent", Strategic Management Journal, 6, pp. 257-272.

Mintzberg H, (1978), "Patterns in Strategy Formation", Management Science, XXIV, pp. 934-948.

O'Connor, A D, (1993), "Successful Strategic Information Systems Planning", JIS, 3, pp. 71-83.

Peppard J (ed.), (1992), "IT Strategy for Business", Pitman, London, p16.

PC Business Software (UK), (1992), MORI Survey, Vol 17, No. 4, pp. 11-13.

Porter M E, (1987), "The Economist, 23/5/87.

Quinn J B, Mintzberg H and James R M, (1988), "The Strategy Process, Concepts, Contexts, and Cases", Prentice-Hall, Englewood Cliffs, N J.

Quinn J B, (1980), "Strategies For Change : Logical Incrementalism", Sloan Management Review, Fall, pp. 7-21.

Quinn J B, (1989), "Managing Strategic Change" *in* "Readings in Strategic Management", Bowman C and Asch D, MacMillan, London.

Schendel D and Hofer C, (1979), "Strategic Management", Little, Brown, Boston MA.

Scottish Office, (1993), Economics and Statistics Department, Correspondence with author.

Senn, J A, (1992), "The Myths of Strategic Systems: What Defines True Competitive Advantage ?", Journal of IS Management, Summer, pp. 7-12.

Simpson R, (1993), "the Risk" *in* Business Scotland, August 1993.

Stowell F A and West D, (1994), "Client-Led Design: A Systemic Approach to Information System Definition", McGraw-Hill, UK.

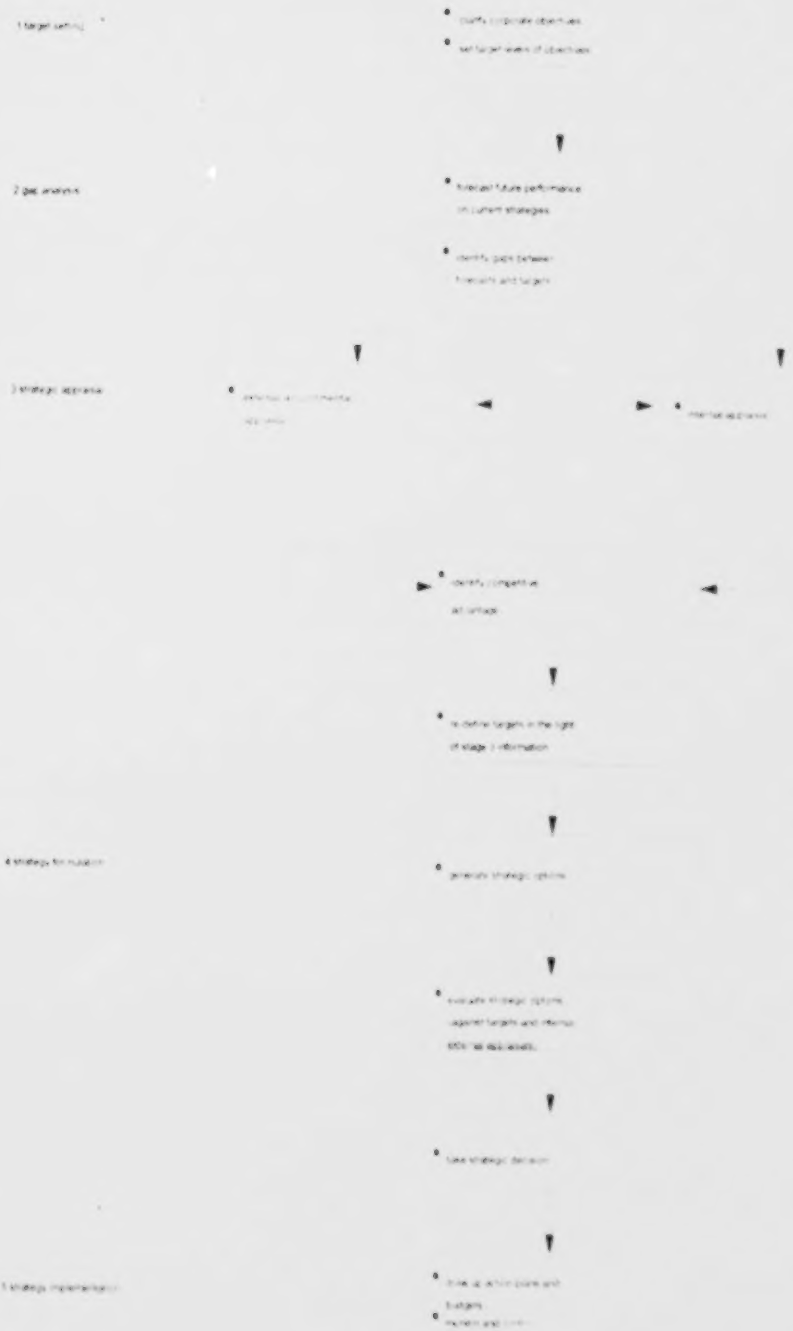
Sutherland E, (1992), "Strategic Management and Information Systems - An Ambiguous Relationship" in Int'l Journal of Information Resource Management, Vol3, No. 2, pp. 24-32, MCB University Press.

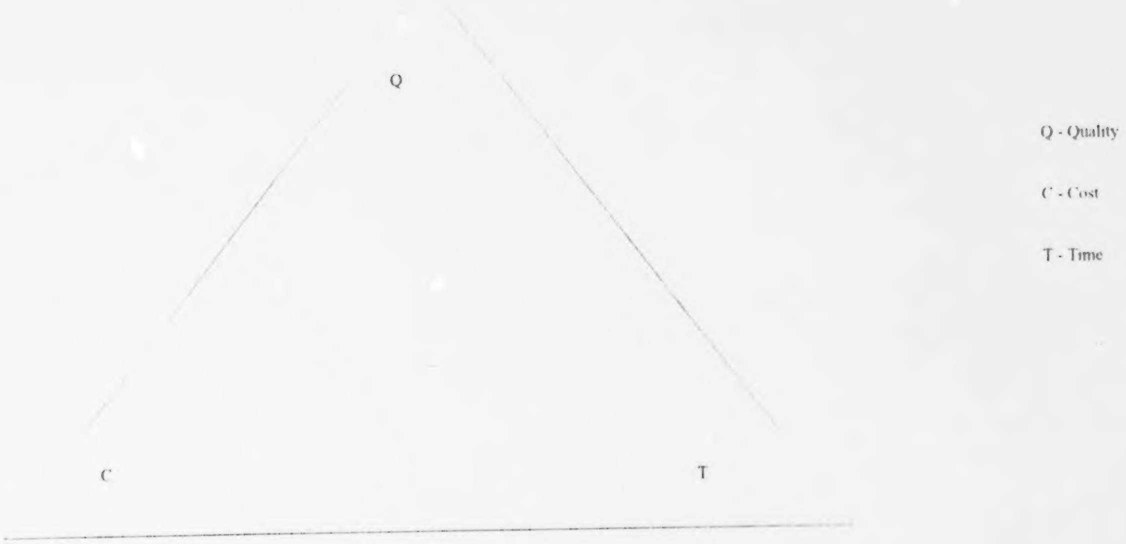
Vickers G, (1983), "An Assessment of the Scientific Merits of Action Research" in Administrative Science Quarterly, 23, pp. 582-603.

Warr A, (1991), PhD Thesis, yet unpublished.

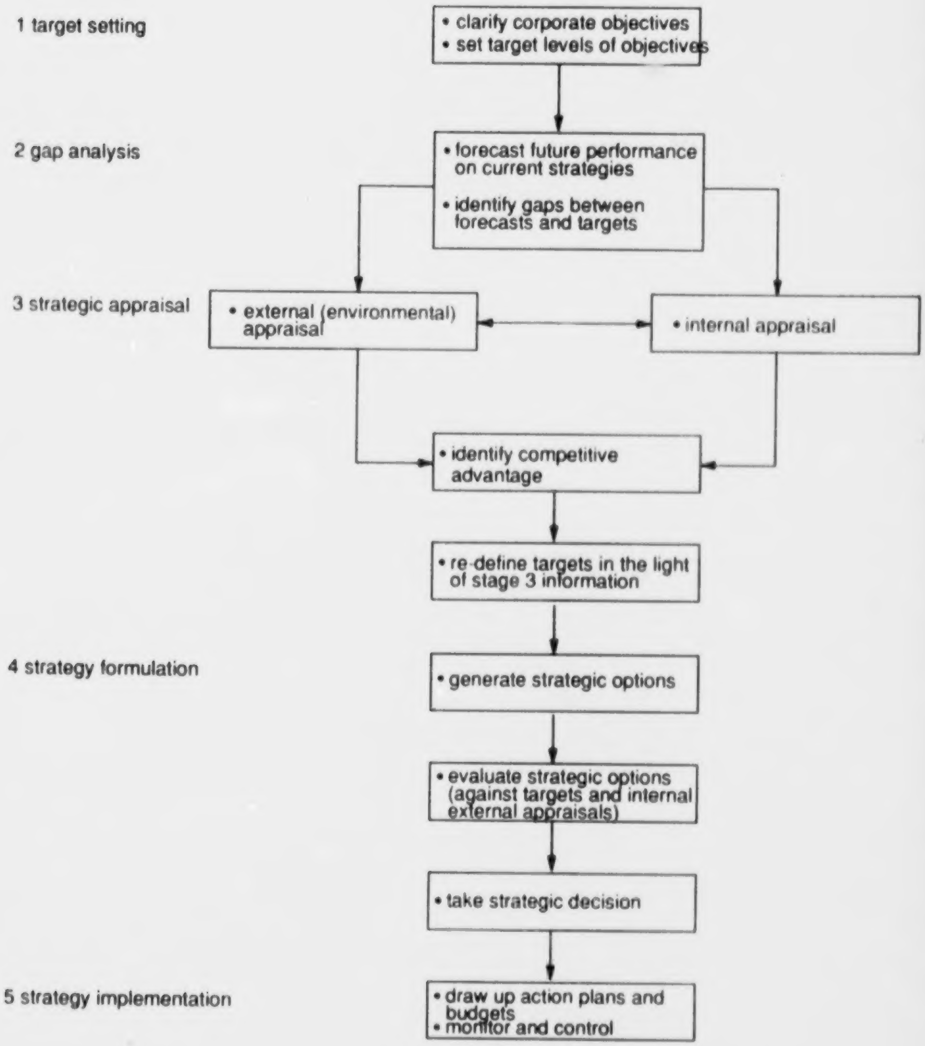
Wooldridge B and Floyd S W, (1985), "Strategic Process Effects on Consensus", Strategic Management Journal, 28, pp. 799-820.

Wooldridge B and Floyd S W, (1990), "The Strategy Process, Middle Management Involvement and Organisational Performance", Strategic Management Journal, Vol 11, 1990, pp. 231-241.


1 target setting
clarify corporate objectives
set target levels of objectives
2 gap analysis
forecast future performance on current strategies
identify gaps between forecasts and targets
3 strategic appraisal
external environmental appraisal
internal appraisal
identify competitive advantage
re-define targets in the light of stage 3 information
4 strategy formulation
generate strategic options
evaluate strategic options against targets and internal/external appraisals
take strategic decision
5 strategy implementation
draw up action plans and budgets
monitor and control


Q
C
T
Q - Quality
C - Cost
T - Time

Quality/Cost/Time Triangle


1 target setting
• clarify corporate objectives
• set target levels of objectives
2 gap analysis
• forecast future performance on current strategies
• identify gaps between forecasts and targets
3 strategic appraisal
• external (environmental) appraisal
• internal appraisal
• identify competitive advantage
• re-define targets in the light of stage 3 information
4 strategy formulation
• generate strategic options
• evaluate strategic options (against targets and internal external appraisals)
• take strategic decision
5 strategy implementation
• draw up action plans and budgets
• monitor and control

Reengineering

ACCESSING THE RICHES OF THE PAST

F.J. Meijer

Centre for Innovative Computer Technology
Grote Bickersstraat 72, University of Amsterdam, Amsterdam, The Netherlands

Abstract

Arrival of a new type of computer often means a definitive end to the life of many interesting programs. To circumvent this demise emulators are often used. They make it possible to let the new computer behave as if it were the old one. Another solution to the problem of accessing old software is to translate the old programs to the language of the new computer. This solution to the problem of portability will be examined. Special attention is paid to error conditions. The notion 'order of errors' is also introduced.

1. INTRODUCTION

The disappearance of the computer language ALGOL is an example of an important accessibility loss. ALGOL was a language constraining its users to certain algorithms[1]. It was useful to solve a large number of problems. To re-access this language is valuable for those interested in ALGOL. Making access possible to interesting old software is the main thrust of the translation - portability - project discussed in this paper.

2. HISTORY OF THE PROBLEM

The importance of the portability problem was recognised from the start. As Fairfield writes "Portability has been an aim of computer scientists virtually since the first computers were invented" (Fairfield, 1985). Some work was done which mainly involved the development of a universal language. All programming languages, including the specific software applications written in such languages, should be translatable to such a language and hence to each other[2].

The first suggestion for a universal language was made in 1958 (Waite, 1977). In that year Strong and others proposed a universal computer-oriented language called UNCOL[3], (Strong, 1958). The aim of the proposed UNCOL-system was to translate programs written in different languages into *one* language: UNCOL-programs. Next, the UNCOL-system would translate these programs into machine language programs for the desired object computer, via a compiler. This so-called two stage rocket never got off the ground.

Over time the tone of the discussion gradually grew increasingly pessimistic. It became more and more apparent that this was a difficult problem. Concerning portability, Mc Carthy calls it "a group exercise in wishful thinking" (Mc Carthy, 1961).

Bemer stated in 1969, that he had not seen a single working program that could translate machine language in the past twelve years (Bemer, 1969). Ward, also in 1969, noted that the method of 'transferability'[4], suggested in 1958, had proved to be insignificant. These failures were not entirely due to lack of effort. There was an increasing awareness that the cost of programming was rising markedly, and could only be reduced by better access to programs on other computers (Ward, 1969). An important part of that cost increment was spent on reprogramming existing software.

The situation improved, somewhat, when it was realised that higher level languages might be of some help. They were introduced around 1970, specifically with the aim of reducing efforts in programming. The problem of porting old software moved to the background. Instead, the discussion centred around how to avoid future problems, and write programs in such a way that they can easily be ported to other computers.

Wallis (1982, a) proposed a number of guidelines programmers should follow to ensure future portability. He proposed such guidelines specifically for the ADA language.

Not only a universal higher language was sought. The same idea - a universal computer language - was tried on the level of assembly languages. In 1986, Fitz and Crockett proposed a universal assembly language. It was possible then to devise such a language as most micro-processors at that time worked internally with registers of the same length[5].

3. DIFFERENT CASES

To port a program various routes may be taken. One route is to translate the instructions of the program of the source computer (M) into the machine language instructions of the object computer (M') during each run of the source program. This is called *emulation*. If a translation can be found which provides the same functions as contained in the original program it is possible to run a program on M' *as if* it were running on M. A large variety of emulators exist. May identifies different generations of emulators[6] (May, Cathy, 1986). Working with an emulator has a number of disadvantages. But, it can be useful as an approximation to the solution of the portability problem. The most significant disadvantage is the sluggishness of the simulated program. This is due, amongst other things, to the proliferation of instructions required by the emulator[7]. A further disadvantage is when recreating an old working environment, the user is forced to work in an unfamiliar software environment[8].

Another route concentrated upon the idea of code disassembly. This notion implies that machine code on M is disassembled and reassembled into machine code for M'. Code disassembly is a means of producing a raw textual version of a program that itself is only available in machine code. After eventual adaptations of the raw output of the disassembler, the code can be translated into code for the new machine.

Still another route is *translation* on at a higher language level. However, this method is not addressed here because there is ample literature available to interested researchers.

The portability problem and its solutions are not solely dependent on the type of computers. The solutions also depend also on the language of the program to be translated. It is rational to make a distinction of the software's language to be translated. Three cases can be discerned according to the language of the code - higher level language, assembler, or machine language:

Case A

The source program is available in a higher level language. If on the object computer a translator is available which translates the source code in instructions to be executed by the object computer there is no basic problem. However, manual adjustments may be necessary because:

1) Specific hardware aspects of the object computer must be accounted for.
2) The interpreters or compilers on the source and object computer assign different meanings to the same source code. In computer jargon these different versions are called "dialects".

Case B

A source version of the program is available in assembler. The translation of the program written for M into code for M' causes no special problems. But, there must be a translator which converts the source code of M into code for M'. Manual adjustments may be needed as mentioned in Case A.

Case C

The program is only available in machine language. This case presents the most problems. An attempt is made to show that these problems can be solved for specific combinations of source and object computer. Programs *can* be written which generate correct code that can be unambiguously executed by the object computer.

In the past researchers have given little attention to Case C. Its possible solutions will be discussed further.

4. POSSIBLE SOLUTIONS; ADVANTAGES AND DISADVANTAGES (CASE C).

If a program written for M can be simulated on M', porting such a program from M to M' and execution on M' is possible. Several solutions to the portability problem were explored:

Solution one.

Every instruction from the source program (p) is translated to a number of instructions for the object computer (M')

Each series of instructions which unambiguously form the translation of an instruction of p is labelled. The same applies to data of p. These labels are used by the portability package (PP) to make correct relations between them.

There are a number of subroutines to deal with the difference in the registers of the source and object computer. Thus, an unavoidable overhead is connected with the execution of ported programs. The large number of instructions generated, far more than the original program contained, is a disadvantage. It makes the output of the PP difficult to recognise as a translation of the original. Furthermore, elaborate bookkeeping is needed to manage all code and data needed by the PP.

Solution two.

Every instruction in the source program is translated to a subroutine-call for the object computer program.

A strict separation between instructions and data is maintained. The area originally used by the source computer to execute the program is treated as a region containing only data of the object program.

The disadvantage of this method, having to call a subroutine each time an instruction of the source program is executed, is relatively insignificant, when compared to its advantages. The high speed of today's computers far exceeds those of yesteryear. Therefore, using many subroutine calls has little negative impact on the execution of the translated program. One of the main advantages is a one-to-one relation between the source and object program instructions. The names of the subroutines to be executed are chosen in such a way that the user of the PP can easily understand which instructions originally were executed by the source computer. Making a separation between instructions to be executed (operators) and data they manipulate (operands) eliminates the problem of running into data. Thus, the possibility of a program crash has been avoided, because one part of working storage will contain only instructions to be executed, and the other the data to be manipulated. This separation has been highly recommended in computer literature, but is seldom found in practice. By maintaining a separate data memory, each datum has to be located. This results in an increase in the number of instructions to be executed, but, safety has its price.

Solution three.

Every instruction in the source program is translated in a statement in a higher language. Executable code is generated by an appropriate compiler or executed by an interpreter.

Computer literature recommends the use of high level languages to solve problems with the aid of a computer. But, in the case of code generation it is not suitable. Usually programs to be translated contain jumps. Furthermore, jumping in and out of subroutines often can be found in these old programs. High level languages either forbid or discourage the use of jumps. Translation of such constructs in a higher language leads to error messages and rejection of code generation. The use of high level languages is applicable *only* if the source code obeys strict programming rules. This method of generating high level statements was not considered as useful.

The second solution is preferred because the relation between the original program and the translation, as well as the segregation between instructions and data, are maintained.

5. THE AMBIGUITY PROBLEM AND ITS SOLUTION

Operating on entities, called data, is the essence of computer operations. The memory of the computer contains not only these data, but also instructions for their manipulation. Both the data to be manipulated and the instructions are stored in files for future use. Unfortunately, it is not possible simply to decide whether the strings of bits in such a file are data or instructions. So the question is, which bit strings are instructions and which are operands? Horspool and Marovac and Yoo state, that complete "translation back" of the program is impossible because instructions and data are stored in the same space, without any indication of what is an instruction or datum. (Opler, 1962)

There is still another problem. It is not clear, beforehand, which data are operands influencing the outcome of calculations and which influence the flow of the program. This ambiguity is the main problem in porting machine code. To make this difference clearer, a distinction is made between data referencing solutions to the problem, and data influencing the flow of the program. The former will be referred to as 'problem-data' and the latter 'control-data'.

The situation becomes more complicated when a datum is used for both functions. For some writers this situation creates an insurmountable difficulty. The next section discusses a possible solution to these problems.

6. SOLUTION: STATIC OR DYNAMIC RESEARCH

Most efforts to overcome the ambiguity problem stem from static research of bit strings. An advantage of this method is that all bit strings which make up the program can be investigated. But often it fails to correctly distinguish between instructions and data. With dynamic research, this is not the case because the program flow is traced which makes it possible to make this distinction unambiguously. The PP, using dynamic research, stores the information of only executed instructions in a file.

Dynamic research makes use of an emulator running on the object computer. This is done because it is not certain if a source computer is available. It may be defective, or the program was written for a hypothetical computer.

A simplistic solution would be to consider the set of instructions which has been formed during the process of emulation as a translation. But, the goal is to generate instructions which can be executed directly on the object computer without interpreting. Simulation is considered, merely, as an intermediary step.

A disadvantage of dynamic research is the possibility that not all instructions of the source program have been executed. There are two reasons for this condition:

1) The program may contain branches for which all possible conditions are not fulfilled. The solution to this problem is to force the execution of branches. Whether these instructions would ever be executed without such a manipulation is not important. At this stage it is only necessary to know whether the forced branch can be executed, and if it consists of instructions.

2) The program may contain superfluous instructions. It is necessary to identify instructions which will not be executed, and unused data. To resolve this problem a separate utility program has been developed which identifies these.

Section 7 and 8 discuss problems that may be encountered if the program to be translated contains bugs.

7. CORRECTNESS OF MACHINE CODE PROGRAMS

Chae Woo Yoo states that the program to be ported should contain as few bugs as possible (Chae Woo Yoo, 1985). The PP's main task is translation and not detecting bugs. If it finds bugs it is an added bonus.

Horspool and Marovac (Horspool, 1980) formulate a number of conditions which a translatable program should fulfil. These are:

1) It contains no self modifying code that changes operation codes or operands of branch instructions.
2) Branch addresses are not dynamically computed, but appear explicitly in the operand fields of branch instructions.
3) Conditional branch instructions are not used where the flow of control is unconditional.
4) Subroutines return control to their calling points, not to their calling points plus some number of bytes in order to skip over parameter lists.

If any of the above conditions are encountered by the PP manual intervention is required. Condition three does not pose a major problem in using the PP as it addresses the correctness of computer programs.

8. ORDERS OF ERRORS

Originally, literature was concerned mainly with the technical criteria for optimizing the use of computers. This is not surprising. When a new tool is developed attention is first directed to its technical aspects and tries to improve upon them. Only if there is a reasonably reliable support system will attention be directed to other aspects.

People need rules to survive. Each profession has its own set of rules. Soon it became clear that the developing programming profession also needed them. Such as: how a program should be written. Most attention has been directed to the writing of programs in higher level languages. The same rules are applicable to programs in assembler. Adhering to them shortens the writing as well as the debugging of computer programs. The discussion about the GOTO-instruction illustrates that conflict can sometimes result as a side effect during the emergence of a new language. (Dijkstra,1968, a) Even with these GOTO-instructions, programs have been written which were reliable and functionally faithful. But sometimes such programs spelled disaster: particularly if they had to be modified.

It is advisable, therefore, to introduce the notion 'the architecture of correct programs'. An example of a program that may be bug free but its architecture is faulty is one containing the following statements:

```
C:=1
if C=0 then...
```

It is obvious that the branch after 'then' will never be executed because "C" has been given the value of 1.

The distinction between bug free and architecturally correct is not used in literature. Dijkstra, for example, expounds the notion of program correctness without making such a distinction (Dijkstra, 1968, b). If an error is encountered, be it on the technical or the logical level, the cause of the error is called a bug. It is advisable to make the distinction 'bug free' and 'architecturally correct'.

In verifying the program the output is incorrect. This faulty output is produced although the algorithm used seems to be correct.

There are different possibilities which may cause this to occur. They are:

1) The application program is incorrect. This is called a first order error. The error occurs on the highest level of programming.

2) The application program is correct, but the software used in developing it, is not. This is called a second order error. The PP may come to a halt if it confronts an insurmountable difficulty. If it continues it may produce incorrect output. In this case modification of the PP may be necessary.

3) There is still the possibility that the application and the higher level language program are correct but the software, on lower levels, contains bugs. These are called third order errors. Such errors exist, but discussion of them would deviate from the main objective of this paper.

So, if a problem is encountered during software development it is important to determine the order of the error. If it is determined that the faulty output is caused by a third order error there is little that the user can do other than seek external assistance from the originator of the supporting software. If a second order error is identified, the user must modify the supporting software. This is only possible if the source code of the supporting software is available. Porting may seem not to be fruitful if a program contains first order errors. But, for archaeological reasons, it may be worthwhile to translate. Furthermore, as the source code of the program during the translation becomes available first order errors can be removed.

During the development of the portability package errors could be attributed to erroneous steps. The burden of deciding the order of an error rests on the shoulders of the user. It is the user's responsibility to master the ability to decide the type of errors encountered and overcome them.

9. CONCLUSION

The solution to the portability problem requires the resolution of a number of associated problems. The more the working environment of the source computer differs from that of the object computer, the more difficult it is to solve the portability problem. This can be mitigated by choosing a source and object computer that use comparable operating systems.

10. BIBLIOGRAPHY

Bemer, R. W.: Program transferability, in *AFIPS conference proceedings vol 34*. Spring Joint computer conference, 1969, p. 606.

Brown, P. J. (ed.): *Software Portability - An Advanced Course*, Cambridge University Press, Cambridge, England, 1955.

Cathey, J.: "An 8080 Simulator for the MC 68000", in *Dr. Dobb's Journal* 11, Jan. 1986, pp 56-82

Chae Woo Yoo, "An approach to the transportation of computer software, in *Information Processing Letters* 21, 1985, pp 153-155.

Coleman, S. S.: "*Janus: A universal intermediate language*", Ph.D. Thesis University of Colorado, 1955.

Dijkstra, E. W. Go To Statement Considered Harmful, in *Com. of the ACM* Vol 11, no 3, March 1968

Dijkstra, E. W. A constructive approach to the problem of program correctness, in *Bitroc.* No. 8, 3, 1968, pp 154-186.

Fairfield, P.: "*The Portability of Low Level Software*", Ph.D. thesis, Dept of SCM, University of Liverpool 1983.

Fairfield, P.: "STRAPS - A Software TRAnsPort System for Low-Level Software, in *The Journal of Systems and Software* 5, 1985, pp 291-302.

Fitz, R.M. and L. Crockett "Universal assembly language" TAB books, Blue ridge summit, P.A. 1986.

Horspool, R. N., and Marovac, N.: "An Approach to the Problem of Detranslation of Computer Programs, " *Computer Journal 23*, no 3, 1980, pp 223-229.

May Cathy: "Mimic: a fast system/350 simulator", in Com. *of the ACM* Vol 6, no 1, May 1985, pp 1-13.

Mc Carthy, J: A basis for a mathematical theory of computation, in *Proc. Western Joint Comp. Conf*, 1961, pp 225-238.

Opler, A., Farbman D. et al: Automatic Translation of programs from one computer to another, *IFIP-Congress 62*, Munich, Germany, Aug 25- Sept 1.

Strong, J. (ed). O. Mock, T. Olsztyn, T. Steel, A. Tritter and J. Weigstein: "The Problem of Programming Communication with Changing Machines: A Proposed Solution, " *CACM 1*, no. 8 (12-18); no. 9 (9-15), 1958.

Tanenbaum, A. S., P. Klint, and W. Bohm: "Guidelines for Software Portability, " in *Software - Practise and Experience 8*, no 6, 1958, pp. 681- 698.

Tanenbaum, A. S., P. Klint, Staveren, van H, Keizer, E.G. and Stevenson J.W: "A practical toolkit for making portable compilers", in *Com. of the ACM* Vol 26, no 9, Sept 1983, p 654-660.

Waite, W. M.: "Janus" in P. J.Brown (ed.), *Software Portability - An Advanced Course*, Cambridge University Press, Cambridge, England, 1955, pp. 255-290.

Waite, W. M.: "Building a mobile programming system" in P. J.Brown (ed.), *Computer Journal 13, 1*, 1950, pp. 28-31.

Wallis, P. J. L.: "The preparation of guidelines for portable programming in high-level languages" in *Computer Journal* Vol 25 no3 Aug 1982. pp 375-378.

Wallis, P. J. L.: *Portable Programming*, Macmillan, London, 1982.

Walraet, B.: *Programming, the Impossible Challenge*, Elsevier Science Publishers. BV., 1989.

Yoo, C. W.: "An Approach to the Transportation of Computer Software, in *Information Processing Letters* 21, 1985, pp 153-155.

---

[1] An algorithm is the description of a pattern of behaviour, expressed in terms of a well-understood, finite repertoire of named ... actions of which it is assumed a priori that they can be done. (Walraet, p. 92)

---

2 Coleman, Poole and Waite proposed JANUS as universal language. In 1975 Druseikis developed SiU2, a language for a hypothetical machine which converted code for the object computer. Tanenbaum considered in 1976 System/370 machine code as a universal 'between language'. (cited in Waite)

3 According to Conway a universal language had been proposed by experts in the field as long ago as 1954.

4 Transferability: the movement of bit strings to another computer.

5 These registers normally were 8-bits.

6 Cathy May uses the term simulator. An emulator is a program that simulates a piece of computer hard ware. Currently the notion 'simulator' is generally used for a program that simulates other systems. (e.g. weather forecasting programs, flight simulators)

7 This is true for contemporary machines. In the future technological advancements may make this argument untrue, assuming that emulators are available for obsolete machines.

8 If a user, for archaeological reasons, is interested in the old working environment, an emulator may be of some interest.

401

G.O.A.T : GENERIC OBJECT-ORIENTED ANALYSIS TOOL FOR INTELLIGENT OBJECT EXTRACTION

Gillian Sleith, Don McFall and John Hughes
Dept. of Information Systems, Faculty of Informatics, University of Ulster,
Shore Road, Newtownabbey, Co. Antrim BT37 0QB, N.Ireland, UK

Abstract

Reverse engineering involves the extraction of design elements from an existing system to produce higher level abstractions. An overview of the TREC (Tool for Reverse Engineering Code) architectural framework, which uses a combination of artificial neural networks and a knowledge-based system to extract objects from imperative code and it's accompanying design documentation is given. The system examines the original source code and deploys a variety of original techniques to gain a detailed understanding of the systems functionality. Existing documentation, stored in a repository, provides additional knowledge about the application domain and is used during a verification process to confirm the existence of previously identified objects. An equivalent object-oriented specification is generated, detailing object classes and their invocations. It is the knowledge-base component and the verification mechanism of TREC that this particular paper focuses on.

INTRODUCTION

With the advent of the object-oriented paradigm, many software developers are keen to evolve existing systems to encompass the many benefits that it offers (Korson and McGregor, 1990; Wirfs-Brock and Johnson, 1990). However as the majority of existing software is written in imperative code, the process of extracting objects and their associated methods, from the code, is proving to be a difficult task. The main reason being that object-oriented applications are based on the concept of procedures being intrinsically linked to data elements, as opposed to the structured ideology were both data and programming logic are 'tangled' together. In order to identify objects, it is necessary to identify useful entanglement in the existing system (Kozaczynski and Wilde, 1992).

Several attempts have been made to develop methodologies and tools to assist with the extraction of objects from structured code (Bennett, 1991) however many of these techniques have proved inadequate. A large percentage of these techniques concentrate on data, as opposed to procedural re-engineering because this process is relatively easy to automate as what you seek to extract is explicit in the code (Harandi and Ning, 1990; Hevner and Linger, 1989). Successive transformations of data model in the forward engineering life-cycle are better defined than for process models. Of all the existing techniques each has their own shortcomings. These include the use of unstructured code as a source of input (Ong and Tsai, 1993), deficit of concrete methodologies (Jacobson and Lindstrom, 1991), scarcity of verification mechanisms to ascertain the correctness of the equivalent object-oriented representation and to authenticate the extracted objects and the lack of utilisation of design documents and domain knowledge (Bowen et al, 1991; Breuer and Lano, 1991).

To meet their true potential, CASE must be applied to the problems of software maintenance by providing standard procedures for reverse engineering and must be workable in both directions i.e. they must provide automated support for both forward and reverse engineering. Future CASE tools must be capable of capturing representations fo old code and hence generating required information from this source. As mentioned previously, data re-engineering will not present any real problems, however procedural reverse engineering can never be fully automated and must allow analyst interaction to identify or provide missing information. By embedding expert systems into reverse engineering tools, system developers will be one step closer to solving the maintenance problem. SOFTMAN was developed by Choi and Scacchi (1991) is one such endeavour to address this problem, as is TREC which attempts to incorporate a rigorous methodology for the reverse engineering process into it's toolset and encompasses :-

- Provision of a rigorous methodology for effective reverse engineering
- Movement to an OO representation to facilitate reuse
- Support for non-CASE systems
- Provision of a verification mechanism to authenticate identified objects
- Construction of object/class hierarchies
- Utilisation of application domain knowledge and human interaction

Towards an Object-Oriented Form : System Overview

A combination of Artificial Neural Networks (ANNs) and a knowledge based system (KBS) within the context of a mapper algorithm, is applied to this problem (McFall *et al*, 1993). This approach draws upon the recommendations in SCORE/RM for the abstraction stage of reverse engineering (Smythe *et al*, 1990). In essence this approach can be viewed in the context of a six layer model (Fig.1). The COBOL source code is pre-processed into a structured form, containing only single entry-single exit procedures (Arnold, 1989). This mapping is achieved using restructuring tools currently available in the market place. The output of this mapping forms the foundation, layer 1, of our model.

The initial step in the algorithm (Mapping 1..2) involves the assembly of conceptual models (Data Flow Diagrams and Extended Entity Relationship Diagrams) from the program's immediate environment. It is often the case when re-engineering a program, that such models are incomplete, or worse, non-existent. Currently available products (Oman, 1990) can be utilised to reconstruct conceptual models if necessary. The advice and expertise of those familiar with the software is also required. These representations are stored in a repository to form layer 2 of our model. Layer 2 is the starting data for the verification process.

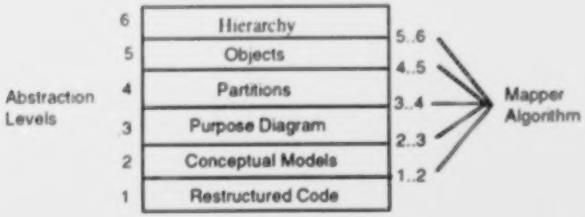


Fig 1. Six Layer Reverse Engineering Model.

Step 2 (Mapping 1..3) examines the source code at layer 1 and constructs from this a dependency diagram (Ballance *et al*, 1990; Ottenstein and Ottenstein, 1984), which is augmented by a verb pattern to produce a representation known as a Purpose Diagram

(PD). This PD is then translated into a representation that will be input to the first ANN, a Hopfield net (Hecht-Nielsen, 1990), the function of which is to optimise the PD.

The output of the ANN will be a graphical form; a reorganised version of the input. Evidence suggests (Hausier *et al*, 1990; Rugaber *et al*, 1990; Rich and Wills, 1990) that the combination of commands used for computer-oriented problem solving in the business application domain form distinctive patterns for each well defined function e.g. a bubble sort, in a source language a well defined shape exists. Function extraction through shape and access analysis, is then applied to the output of the neural network at Step 3 (Mapping 3..4..5) by assigning each verb in COBOL a movement in a specific dimension, returning a functional description of a procedure, to facilitate the binding of that procedure with a specific data entity. This is done using a second neural net, a Hamming net (Lippmann, 1987) for pattern recognition. The input to the ANN is formatted as a matrix, which consists of program line numbers on the vertical axis and program commands across the horizontal axis. For each line of code in the procedure being analysed, whenever a command is encountered, then a mark is placed at the appropriate position on the grid.

The output of the algorithm is a series of ADTs, composed of data elements with corresponding functional labels. Each functional label represents functionality associated with one particular data element. Pseudo code describing this functionality is generated and stored with each functional label. Step 4 (Mapping 5..6) involves the construction of an object hierarchy from these pooled ADTs. This is then fed into the KBS, which will already contain knowledge about the application domain. The knowledge stored in the KBS is extracted from existing documentation and stored as conceptual models, at layer 2. This layer 2 knowledge and the objects produced by the ANN, for layer 5, are used together in a verification process.

MAPPING 2..5 : OBJECT EXTRACTION FROM CONCEPTUAL MODELS

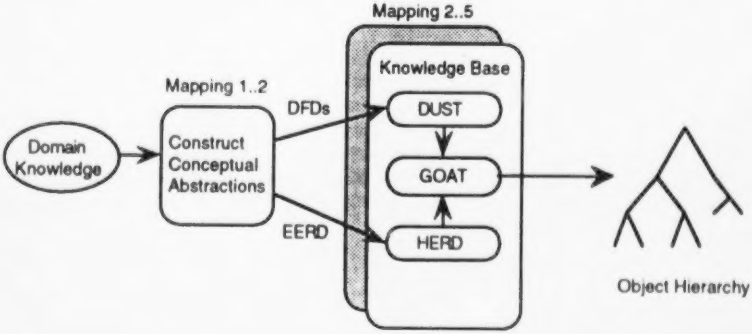


Fig 2. Knowledge base component of TREC architecture, responsible for the extraction of objects from design documentation.

In this stage of the algorithm, a movement is made from the second level of abstraction, in our 6 layer model, to the fifth (Fig 2). The conceptual models under consideration are Extended Entity Relationship Diagrams (EERDs) (Chen, 1976; Teorey *et al*, 1986) and Data Flow Diagrams (DFDs) (Gane and Sarson, 1979). The EERD is used to identify the various object classes and their properties in the application domain and to indicate potential associations between these classes. The DFDs on the other hand, are used to extract the functionality that is associated with the previously identified objects i.e. to show system behaviour. They highlight the functions, mappings, constraints and functional dependencies within the system. Once operations have been identified, they are placed in the appropriate object class. It is also possible to identify communication between the object classes. All levels of DFDs are stored for each program in the application domain and the system then determines which to use (Cordes and Carver, 1988) and (Sangbum and Carver, 1991). The conversion component GOAT (Generic Object-oriented Analysis Tool) provides the automated support for our methodology, which lays down guidelines for the conversion of an EERD into an object model and also provides rules to aid in the extraction of functionality from within DFDs. However the methodology cannot be fully automated and requires a certain degree of human interaction to provide additional or missing information to aid in its analysis process.

The object model that we aim to construct is based upon that used in the OMT methodology (Rumbaugh *et al*, 1991). This formal model acts as the basis for our re-engineered system and represents the various objects and relationships within the system. It is used to represent the static, structural and 'data' aspects of the system - showing the objects and their properties, the operations associated with each object class and the relationships between the objects. OMT consists of two other system models; the dynamic model and the functional model. The dynamic model represents the temporal, behavioural and control aspects of the system and the functional model is used for specifying the transformations of object values and the constraints on these transformations; these are represented by DFDs. Table 1 illustrates the sources within the algorithm which produce the necessary information for construction of these three models.

Information Required	**Algorithm Source**		
	[3..4..5]	[5..6]	[2..5]
Object Model			
Class names	x		x
Data attributes	x	x	
Associative relationships		x	x
Aggregate relationships		x	x
Inheritance relationships		x	x
Dynamic Model			
Class names	x		x
Finite set of states for class			
Events causing state changes			
Activities linked to states			
Actions linked to state changes			
Events depicting class co-operation			
Sources of external stimulus			
Functional Model			
Internal/External sources/sinks			x
Data attributes		x	
Activities		x	x
Actions		x	x

Table 1. Source of information for representation of system in OMT.

RULES FOR CONVERTING EERD INTO OMT OBJECT MODEL

Rule 1 : Each entity becomes an object class in the Object Model.

Rule 2 : All attributes for a particular entity correspond to properties of the new class.

Rule 3 : Each relationship indicates a message connection or association between corresponding objects.

Rule 4 : Relationships are typically represented by properties within an existing class (Hughes, 1991) (Fig. 3) e.g.

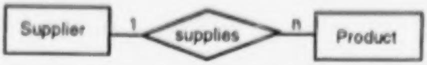


Fig 3. Conversion of EERD relationships into corresponding object properties

```
class  Supplier
            properties
            supplier-code        : string;
            name                 : string;
            address              : string;
            phone-no             : integer;
            products             : Set (Product) inverse is Product.supplied-by;

class  Product
            properties
            product-code         : string;
            description          : string;
            size                 : string;
            retail-price         : money;
            cost-price           : money;
            supplied-by          : Supplier inverse is Supplier.products;
```

The relationship becomes a new property within each class and assumes a type corresponding to name of the participating entity, prefixed with the words "Set of" if more than one entity can participate in the relationship. This representation provides the ability to define properties of entities in terms of other entities.

Rule 5 : N:M relationships can be implemented as an association or as a separate class. If there are attributes that naturally belong to the link, then it will be best to model the relationship as an association. This can be done by flagging those attributes of an entity which belong to the link. The tool will then reassign the attributes accordingly.

Rule 6 : "Is_A" relationships indicate inheritance within the Object Model. A superclass, in the EERD, becomes a generic class in the Object Model and subclasses become specialisations of the generic class.

RULES FOR EXTRACTING OF FUNCTIONALITY FROM DFDS

During this phase, it is important that only *logical* DFDs should be considered. With logical DFDs, there is a 1:1 mapping between data stores and entities, but this does not hold true for physical DFDs.

Each data store corresponds to an object, whilst external entities form external stimuli to the system. A number of the objects identified from the DFDs will be duplicates of those identified from the EERD. It is therefore necessary to create a cross-referencing table of multiple names used to identify a particular object.

Before extraction of methods can begin, it is necessary to establish which of the DFDs to work with, as not all levels of DFDs will be relevant. Only the lowest level DFDs, for each level 1 process, should be utilised. Consider the example below, of a complete set of DFDs representing a system. Initially the DFDs consists of the following processes:

Level 1	1.0, 2.0, 3.0, 4.0
Level 2	1.1, 1.2, 1.3, 1.4, 2.1, 2.2, 3.1, 3.2, 4.1, 4.2, 4.3, 4.4, 4.5
Level 3	2.1.1, 2.1.2, 2.1.3, 2.1.4, 2.2.1, 2.2.2
Level 4	2.1.3.1, 2.1.3.2

However after selection only the following processes are identified for utilisation in the conversion process, to represent the system, in an object-oriented environment :-

Process 1	1.1, 1.2, 1.3, 1.4
Process 2	2.1.1, 2.1.2, 2.1.3.1, 2.1.3.2, 2.1.4, 2.2.1, 2.2.2
Process 3	3.1, 3.2
Process 4	4.1, 4.2, 4.3, 4.4, 4.5

Each DFD process selected corresponds to a method in the new system. Methods can be public, in the interface for users to reference directly, or private/internal, when they cannot be directly accessed by other objects or the user. The data flows between processes indicate method calls between objects or within objects. The data items contained in these flows may be an indication of the arguments for the method invocations.

Rule 1 : A process that receives an incoming data flow from an external entity or data store is an public/interface method.

Rule 2 : Processes belong to the stores that they access. If a process accesses more than one store, then the process must be decomposed further, into a number of sub-processes, which are split between the objects involved. It is then necessary to examine alternative documentation to determine which of the data stores is accessed first. The operation will reside in this store and it will, in turn, send message calls to the appropriate objects that house the other methods involved in the process.

Rule 3 : If several serial processes have incoming data flows from the same store, then the first process to accept the flow is the interface method for that particular operation.

Rule 4 : Starting at the interface method, identify chains of methods, for an operation, by following the data flows between processes.

An operation starts at the public/interface method and terminates with a file update i.e. when data flows from a process and into a data store or external entity. The operation consists of the intervening methods.

If the terminating process sends data to subsequent processes which, in turn, access a different store, then these processes become methods within that second object. The process immediately following the terminating process becomes the interface method for the operation belonging to the new object. The data flow from the terminating method to the interface method, for the second object, is an example of a message being sent to a different object, instructing it to initiate one of its operations.

Rule 5 : If data flows from a process X, to a process Y, where the processes belong to the objects A and B respectively, then the receiving process Y, must be in the interface of object B.

Rule 6 : If data flows into a store, then this flow indicates that an *UPDATE* operation has taken place. Update operations are those that modify, delete or add data.

Rule 7 : Each flow out of a data store and into a process, is either a 'query' or 'read' operation, which has no side-effects on the data store object i.e. the data store is not subject to any changes.

SYSTEM OUTPUT

The output generated by the system will be an object-oriented description of the original COBOL program and will consist of class specifications, of the following format :-

```
class <class-name>
        inherit <class-name>
        properties
        attribute1:attribute-type;
        attribute2:attribute-type;
        operations
                public
                        op1(arg1, arg2,..);
                        op2(..);
                private
end <class-name>
```

These specifications will be supplemented with documentation relating to each method, incorporating details of method invocations and their associated functionality.

VERIFICATION MECHANISM

This is used to compare the two sets of output generated by TREC, which comprises of a hierarchy of ADTs produced from the original structured code and a second object hierarchy extracted from the program's design documentation using GOAT. The verification mechanism is a two-pronged approach, consisting of two distinct algorithms :-

- Correlation Algorithm
- Domain Matching Algorithm

Both of these algorithms accept the two hierarchies as input. The purpose of this mechanism is to verify the authenticity of the identified objects i.e. identification of the same object from both sources will establish that object's definitive existence within the application. It may also highlight any discrepancies between the original design documentation and the code i.e. does the program provide all the processing capabilities detailed in the design material? Since these algorithms are independent of one another it is not necessary to perform both to authenticate the objects, however a more accurate result will be produced if both are deployed in the verification process.

The Correlation Algorithm provides a purely mechanical method for statistically matching the properties and operations of an object in one hierarchy with those of an object in the other hierarchy. Each object property and operation is assigned a numerical value, the results of which are plotted on a graph. Pythagoras Theorem is used to determine the 'nearest neighbour' to the original object under examination. This technique involves a large amount of personal judgement on the software engineer's part and is very much open to individual perception and interpretation of the system.

The Domain Matching Algorithm uses repertory grids to provide additional semantic information about the application domain that is not directly available from the program code and it's accompanying documentation. Associations between objects are also addressed in this approach, which the correlation algorithm fails to do. The repertory grids from the various application domains are compared to provide an overall % match. This technique can also be used to aid in the storage and retrieval of reusable components from within a CASE repository.

CONCLUSION

This approach describes a number of new techniques which are being applied to the problem of re-engineering. Several sub-system prototypes have been implemented and are currently being evaluated with regard to several case studies of existing information processing systems. It is not yet clear to what extent we will be able to automate the system, but we know it is not possible to fully automate this approach as a certain amount of human judgement is required to establish the authenticity of the output generated by the system.

The main problems encountered, to date, have been the ability to construct object hierarchies from the identified objects and also the extraction of parameters for the invocation of methods. This inability to identify parameters may be due to the choice of programming language, as all data is global in COBOL. However the tool kit should ease some of the burden on the software engineer and at the very least produce a good object-oriented description of the required system, detailing system objects and their required functionality. This can then act as a basis for forward engineering into an object-oriented system.

REFERENCES

Arnold, R.S. (1989), Software Restructuring, *Proc. IEEE*, Vol.77, No.4, pp.607-617.

Ballance, R.A., Maccabe, A.B., and Ottenstein, K.J., (1990), The program dependence web, *Proc. ACM SIGPLAN Conf. on Programming Language Design and Implementation*, pp.257-271.

Bennett, K.H., (1991), Automated support of software maintenance, *Information and Software Technology*, Vol.33, No.1, pp.74-85.

Bowen, J.P., Breuer, P.T., and Lano, K.C., (1991), The REDO Project : Final Report, *Programming Research Group Technical Report No.PRG-TR-23-91*, University of Oxford, England.

Breuer, P.T., and Lano, K.C., (1991), Creating specifications from code : reverse engineering techniques, *Software Maintenance : Research and Practice*, Vol.3, pp.145-162.

Chen, P.P.S., (1976), The entity relationship model : towards a unified view of data, *ACM Trans. on Database Systems*, No.1, pp.9-36.

Cordes, D.W., and Carver, D.L., (1988), Knowledge base applications within software engineering : a tool for requirements analysis, *Proc. of 1st Int'l Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, June, pp.267-272.

Gane, C., and Sarson, T., (1979), "Structured Systems Analysis : Tools and Techniques", Prentice-Hall, NY.

Harandi, M.T., and Ning, J.Q., (1990), Knowledge-based program analysis, *IEEE Software*, January Issue, pp.104-112.

Hausier, P.A., Pleszkoch, M.G., Linger, R.C., and Hevner, A.R., (1990), Using function abstraction to understand program behaviour, *IEEE Software*, January Issue, pp.55-63.

Hecht-Nielsen, R., (1990), "Neurocomputing", Addison-Wesley.

Hughes, J.G., (1991), "Object-Oriented Databases", Prentice-Hall Int'l.

Jacobson, I., and Lindstrom, F., (1991), Re-engineering of old systems to an object-oriented architecture, *Proc. OOPSLA*, pp.340-350.

Korson, T., and McGregor, J.D., (1990), Understanding object-oriented : a unifying paradigm, *Comms. of the ACM* , Vol.33, No.9, pp.40-60.

Kozaczynski, W., and Wilde, N., (1992), On the re-engineering of transaction systems, *Journal of Software Maintenance : Research and Practice*, Vol.4, No.3, pp.143-162.

Lippmann, R.P., (1987), An introduction to computing with neural nets, *IEEE ASSP Magazine*, April Issue, pp.4-22.

McFall, D., Sleith, G., and Hughes, J., (1993), Reverse engineering structured code to an object-oriented representation, *5th Int'l Conf. on Software Engineering and Knowledge Engineering*, pp.86-93.

Oman, P., (1990), Maintenance tools, *IEEE Software*, May Issue, pp.59-65.

Ong, C. L., and Tsai, W. T., (1993), Class and object extraction from imperative code, *Journal of Object-Oriented Programming*, Vol.6, No.1, March/April, pp.58-68.

Ottenstein, K.J., and Ottenstein, L.M., (1984), The program dependence graph in a software development environment, *ACM SIGPLAN Notices*, Vol.19, No.5, pp.177-184.

Rich, C., and Wills, L.M., (1990), Recognizing a program's design : a graph parsing approach, *IEEE Software*, January Issue, pp.82-89.

Rugaber, S., Ornburn, S.B., and Le Blanc, R.J., (1990), Recognizing design decisions in programs, *IEEE Software*, January Issue, pp.46-54.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., (1991), "Object Oriented Modeling and Design", Prentice Hall.

Sangburn, L., and Carver D. L., (1991), Object-oriented analysis and specification : a knowledge base approach, *Journal of Object-Oriented Programming* , January Issue, pp.35-43.

Smythe, C., Colbrook, A., and Darlinson, A., (1990), Software reengineering : the next step?, *USPDI Software Maintenance and Reengineering Conference*, pp.1-7.

Teorey, T.J., Yang, D., and Fry, J.P., (1986), A logical design methodology for relational databases using the extended entity-relationship model, *Computing Surveys*, Vol.18, No.2, pp.197-222.

Wirfs-Brock, R.J., and Johnson, R.E., (1990), Surveying current research in object-oriented design,*Comms of the ACM* , Vol.33, No.9, pp.104-123.

ORGANISING THE REDESIGN PROCESS IN SYSTEM DEVELOPMENT

Kristin Braa, Tone Bratteteig & Leikny Øgrim

University of Oslo, Department of Informatics, POBox 1080 Blindern, 0316 Oslo, Norway.
Email: kbraa@ifi.uio.no tone@ifi.uio.no leikny@ifi.uio.no

ABSTRACT

The paper discusses the notion of redesign, and identifies three dilemmas related to post implementation change of information systems. The dilemmas are 1) system development aims at designing stable computer systems for changing environments, 2) the activities requiring most time and effort are based on the weakest planning and organisation, and 3) despite the volume of user initiated system changes, user participation is not on the agenda in redesign activities. These dilemmas are grounded in literature and case studies. A way of organising redesign: Version Projects, and a participatory technique: Priority Workshops are proposed as means to address the dilemmas.

1. INTRODUCTION

The seven members of the computer department staff in the bank "Safe & Stable", share the responsibility for providing computer support for 120 users every day[1]. The department has a long backlog of user requests, and a very stressing work situation filled with telephone calls from demanding, angry, and even desperate users trying to get the support they need to do their work from the current computer system. The staff knows some of the users quite well, and the users call them directly when they have trouble—not bothering to follow the formal procedures for making requests. The staff also has to provide new computer based functions proposed by the management or the sales department, in order to win new market shares. Most of the time the computer department staff is working with minor adjustments based on informal user requests, without any specific plan or schedule to refer to. They are actually performing redesign in a somewhat unconscious manner. After some years, they lose the overview of the computer systems in "Safe & Stable" and start being uneasy about changing things because they do not know exactly what will happen if they change too much.

---

[1] This example is inspired by several cases from the FIRE project. The FIRE project is described in, eg, Braa et al, (92). Cf. also Bjerknes et al, (91).

For several years, problems concerning maintenance and enhancement of computer based systems, have been reported. Low quality documentation, difficulties with collection and priorities of user requests, competing claims for programmers' working hours, difficulties in keeping time schedules, poor user training, high turnover rate both among users and programmers, are just some problems described in, eg, Lientz & Swanson (80), Lientz & Swanson (81), Bendifallah & Scacchi (87), and Sørgaard (89). The main source of anxiety has been the volume of resources needed for activities labelled maintenance. Literature reports that over 50 per cent of the total IT resources are spent on maintenance of computer systems (Lientz & Swanson, 86; Sørgaard, 89).

Lientz & Swanson (81) claim that work with existing application systems to a large extent consists of continuous development, and that the most important single activity is to offer enhancements to the users. They state as early as in 1981 that the most important research issue is the development of a theoretically founded and practically useful classification of work performed on existing application systems. This paper provides a theoretical discussion about post implementation activities, and suggests an alternative way of reflecting and acting in redesign.

The theoretical discussion is presented through three dilemmas of post implementation activities. The identified dilemmas are 1) that system development aims at designing stable computer systems for changing environments, 2) that the activities requiring most time and effort are based on the weakest planning and organisation, and 3) that despite the volume of user initiated system changes, user participation is not on the agenda in redesign activities. The proposed solutions, Version Projects and Priority Workshops, address these dilemmas and provide ways of handling them.

The paper is based on three different sources. The first one is a literature survey on maintenance, redesign and related areas, carried out as part of the FIRE project in 1991. The study is partly documented in Øgrim (92). The second source is a series of empirical studies, some of which are included in the FIRE project (Bratteteig & Øgrim, 92; Braa et al, 93; Kaasbøll & Øgrim, 94). The studies are conducted as case studies (Bratteteig & Øgrim, 92; Braa et al, 93) or quantitative investigations (Kaasbøll & Øgrim, 94). The technique Priority Workshop is tested in an organisation during the autumn 1993 (Braa, 94).

The paper is structured as follows: In section 2 the concept of redesign is discussed. Section 3 includes a short discussion of three identified dilemmas of post implementation activities. In section 4, the basic ideas for a redesign method is discussed: a proposal for organising redesign and a proposal for a participatory technique for transparent decision making in redesign. Section 5 concludes the paper.

2. REDESIGN: RE- AND -DESIGN

Redesign includes organisational and technical changes of a computer based system after implementation. Redesign denotes system development when one or more components of an existing computer based system will be a part of, or be integrated with, the future system (Braa et al, 92). "Re-" focuses on the existing computer based system, aiming at keeping the experience, the knowledge, and the concrete system modules that can contribute to a new computer based system. "-Design" points to the constructive aspects.

Of course, there exist also other approaches to post-implementation activities. Re-use aims to the use of existing algorithms, modules, and programs in new contexts (Basili, 90; Chikofsky & Cross, 90). Re-use is important for making system development for effective, both in programming and testing. The programmers can avoid reinventing old solutions, concentrating on enhancing the solutions or designing solutions to new problems. Re-use of code is an important technical aspect of redesign.

Reverse engineering denotes the process of deciphering a design from an existing computer system (Chikofsky & Cross, 90; IEEE, 90). This is useful for achieving a technical overview of an existing computer system, for integration with other computer systems, or for moving a computer system from one technical platform to another. A technical overview is necessary in redesign, when deciding which elements of the existing systems to keep, and which parts to change.

Re-engineering starts with a computer system in operation, restores it to some extent, and—if desired—adds some new functionality to the system (Chikofsky & Cross, 90)). Re-engineering may be a part of redesign, if the desired changes are mainly technical, eg, when converting a system to a different platform. Techniques for re-engineering are, however, neither concerned with how computer systems are used, nor with the changing environment of the systems.

Redesign of a computer based system includes all these technical matters. In addition, needs for new functionality caused by organisational changes, user needs, market or other environmental factors, also is included in redesign.

Maintenance can be categorised as corrective, adaptive, or perfective maintenance (Swanson, 76; Lientz et al, 78). Adaptive maintenance includes necessary adjustment of the computer system based on changes in the organisation, market, or technology. Perfective maintenance denotes further development based on user claims or new technological possibilities.

In this paper, we make a distinction between corrective maintenance and redesign. According to the above notions, redesign includes functional changes categorised as adaptive and perfective maintenance.

3. DILEMMAS OF POST IMPLEMENTATION ACTIVITIES

The problems described in the previous sections are rooted in a complex web of organisational and technical matters. In this section, we discuss the three dilemmas we have identified.

3.1. System development aims at designing stable computer systems for changing environments

The very concept of maintenance is based on a view that computer systems are finished products (Bjerknes et al, 91). Maintenance of computer systems is aimed at restoring and repairing in order to preserve the functionality. This product-oriented perspective isolate the computer system from its surroundings in order to preserve stability and control (Floyd, 87). Maintenance of computer systems occurs as ad hoc changes. The need for maintenance can be caused by errors in software or hardware, by changes in the use of the system, in the organisation, in the market, or in the technological environment.

Organisations change continuously: Some changes stem from environmental shifts, eg, changing markets, others occur as results of individual or group behaviour, eg, development of new ways of using computer systems (Gasser, 86). Organisational changes often lead to new requirements for the computer system (Curtis et al, 88). Technological development also encourages changes in the current computer systems. New technology (eg, architecture, user interfaces, software engineering tools) initiates restructuring and replacement of old or old-fashioned systems (Braa et al, 92). The technology itself creates new possibilities and thus legitimates new user claims.

A story from a bank (Bjerknes et al, 91) illustrates the dilemma: viz that system development aims at designing stable computer systems for changing environments:

> The Customer Service System (CSS) was built in the 80ies. The system was well integrated with the traditional bank routines. After about two years of use, the market division decided to offer a new service to its customers. The idea was to give a discount of 1 per cent on the interest rate of all loans to customers who saved more than $ 1,500 a year. A marketing campaign was started and a lot of customers signed on to it. However, the market division forgot to tell the computer department.
>
> The CSS was designed to support a certain definition of customer services. The new service was based on a different definition of customers, loans, and the relations between the two, thus a major change in the CSS was needed in order to support the new service. Over a period of time, the computer department had to work almost exclusively with this single redesign, and they managed to implement the necessary changes in time. They had, however, to use "quick and dirty" solutions to achieve this, creating a "hacker solution", contradicting the existing data structure. By this, new errors were produced—and a lot of extra maintenance work.

Many of the problems of maintenance are caused by an organisational behaviour based on the view that computer systems are stable artifacts. This is contradicted by experience: the resources required for post-implementation activities are substantial. These facts are not yet materialised in changes of organisation and work styles of redesign.

3.2. The activities requiring most time and effort are based on the weakest planning and organisation

A large proposition of resources is invested in post-implementation system activities. As mentioned, literature reports that over 50 per cent of the total IT resources are invested in maintenance of computer systems (Lientz & Swanson, 86; Sørgaard, 89). Nevertheless, neither maintenance nor redesign are addressed as activities in most system development methods (Biskup & Kautz, 90). The discussion about problems in maintenance by Lientz & Swanson (80) concludes that the main problems are organisational rather than technical. It is a paradox that the activities requiring most time and effort are based on a poor allocation of resources for planning and organising activities.

A case study carried out in the FIRE project can be used to illustrate this dilemma. The study was performed during a period of more than two years, and is documented in (Bratteteig & Øgrim, 92; Braa, 94).

> The case site is a Development Firm (DF) which until recently has been controlling the market on systems for wages and personnel for a specific industrial branch.
>
> Approximately 90 per cent of the employees are working with user support, maintenance, and further enhancement of the existing application systems. The Wages and Personnel System (WPS) is delivered in several versions and variants, tailored to different computer platforms. DF delivers up to four versions each year. Poor planning of new versions cause problems for the user organisations that have to handle upgrading and minor changes several times a year. In addition, the versions are not properly tested before distribution—they always contain errors.

DF has no formal procedure for making priorities of user requests. The designers of the WPS state that versioning is a process to catch up every customer's needs. Changes are initiated 50 per cent from the customers and 50 per cent from the designers own fantasy. Strategic plans for conscious development of the WPS do not exist.

It sometimes happen that the next WPS version is initiated and documented by a new customer, who adds an appendix in the user handbook according to their special needs and calls this a requirements specification. This then becomes the specification of the next WPS version which is delivered to all customers. However, this version is made in order to catch a new customer, not to meet user claims. Statements from users as: "A new version is not always experienced as an improvement" and " Is it possible to skip the next two versions?" are then understandable (Braa, 94).

A study of Lientz & Swanson (80) conducted in 487 organisations and a succeeding study by Beath & Swanson (90) support that post implementation activities often are based of weak planning and organisation—still, computer systems are changed. The work is performed as ad hoc maintenance, in between the "real" work tasks, or performed by a special maintenance group. Unplanned change deteriorates both the technical and functional quality of the computer system (Bjerknes et al, 91; Martin & McClure, 83). Unplanned error corrections might lead to so called "spaghetti-code" which is difficult to overview (Lientz & Swanson, 80; Myers, 79). The maintenance staff will be too busy to judge the total quality of the system. In addition, research shows that every error that is corrected may well be a source of new errors (Lientz & Swanson, 81; Espeseth, 92). The planning of what errors to be corrected is thus crucial (Hetzel, 88; Evenshaug, 94).

3.3. Despite the volume of user initiated system changes, user participation is not on the agenda in redesign activities

In Scandinavia, users have a formal right to be represented in project groups in traditional, original design of computer systems. The Working Environment and Workers' Protection Act (Directorate of Labour Inspection, 78) states that those affected by a change to a computer system shall participate in deciding on that change. The General Agreement between the Norwegian Federation of Trade Unions and the Confederation of Norwegian Business and Industry (LO-NHO, 90) gives the local trade unions the right to elect representatives to participate in development projects. However, after the installation of the computer system, the project groups are normally dissolved, and hereby the formal organisation to support the users' influence disappears.

This would not have been a problem if the user requirements were stable throughout the life cycle of the computer based system. This is, however, not the case: user requirements to a computer system increase with experience of using the system (Lientz & Swanson, 81).

We will illustrate this dilemma through empirical cases. During the autumn of 1993, 8 redesign processes in 4 organisations were studied, in order to find the degree of user participation (Kaasbøll & Øgrim, 94).

The study includes organisations from both public and private sectors, and they are all experienced users of computer systems.

> Six of the eight redesign processes are organised as projects. Only 2 of the 6 projects include a trade union participant, as the General Agreement states. Only organisations with strong unions include trade union representatives as project members. In 3 of the cases, some kind of "super-user", usually selected by management, has partly undertaken the union representative's role. The super-user often is responsible for gathering user requests, for making priorities between the requests, and for negotiating changes with either the computer department or the management. Anyhow, the super user is not elected by her fellow workers, and is responsible only vis-a-vis management.

The collection of user requests and changes, and the priorities and decisions concerned with them are left to the judgement of a super-user and a design team of system developers. Users have influence, not as organised interest groups, but as individuals. Factors like status and position in the organisation, technical knowledge, and personal relations to the designers or the super-users are important.

In system design user participation is important because users are the experts on the work the system is supposed to support; the users possess knowledge valuable to the system developers (Briefs et al, 83; Bjerknes et al, 87; Greenbaum & Kyng, 91; Namioka & Schuler, 92). If users are supposed to participate in making design decisions, they must be given insight into the decision process (Braa, 92). Transparency of the decision making process reveals the rationale of decisions and priorities. Thus, transparency in decision making about the future of a computer based system is a necessary condition for user influence. Open discussions and well reasoned decisions may increase the quality of the priorities made, and by this affect the quality of the system (ibid.).

4. ORGANISING REDESIGN

The previous sections have described and discussed problems and reasons for problems at work concerned with existing computer systems. This section suggests solutions based on the problems that the dilemmas address. The solutions are founded on a conception of system development as an evolutionary process, in which the transitions between the changes are planned and supported.

> The in-house development project of the original CSS in "Safe & Stable" consisted of six computer scientists and four users from the major business areas in the bank. The system was declared to be finished after three years, and the project group was dissolved. The responsibility for CSS was turned over to the company's maintenance group. However, several user requests for changes had been postponed due to time limits set by the management. The maintenance group had to complete tasks left over from the development group. As they had not participated in the development process, they lacked knowledge about the system and its foundation, and accordingly with a growing backlog of user claims.

The division of work between the system development and maintenance staff is expressed through the formal organisation of most computer departments. Software work can be organised in different ways, depending on work tasks, application areas or system life cycle (Beath & Swanson, 90). All forms of organisation include some division of work, and thus division of knowledge. All forms of organisation lead to problems with transfer of knowledge between the organisational units. Lientz & Swanson (80) claim that separate maintenance departments increase the efficiency of work. They also claim that the knowledge transfer from maintenance department to development departments is of importance to improve the maintainability of new computer systems.

Quite contradictory, Naur (85) claims that the essence or rationale of a computer system must be understood in order to maintain or enhance it. This essence—or theory, to use Naur's own notion—is held exclusively by those who originally designed the system. In his terms, a division of work according to a system's life cycle is not possible if the system is to perform according to the original ideas.

From our opinion knowledge transfer is possible: if not, maintenance work or even teaching had been waste of time. However, Naur has a point when emphasising that defining design rationale is a complicated problem—transferring it to others even more. In this section we propose ways to handle the problems of knowledge transfer.

4.1. Organising redesign in Version Projects: a combination of line and project

We propose redesign organised according to system versions rather than traditional line organisation in a maintenance group or department. Version Projects organise the system development work as a series of projects, each connected to the (re-)design of a version of the system. Version Projects should utilise characteristics from both line and project organisation.

Daily corrective maintenance, user training, day-to-day user support is conducted by the line organisation. This work emphasises the continuity concerning the computer system and its use. Errors and proposals for changes are logged, and a routine for initial categorisation and priority of user requests should be made.

Larger changes are performed in organised manner, in a Version Project. The responsible line staff initiates every new redesign project using predefined routines for when and how to initiate new projects. The routines could refer to various characteristics of the system and system use, eg, periods of time (every month or half year), amount of requests, degree of importance of proposals or requests, when claimed from management or unions, or environmental changes in market or technology. The actual routine must be tailored to the actual organisation and the systems in use.

Every Version Project aims to build a new version of the system. The Version Project is concerned with the innovative and dynamic aspects of redesign. Every project should include both experienced and novice user representatives in order to avoid the emergence of a permanent, system specific, exclusive culture of "super users" or "system experts" not really representing the users. Continuity between the Version Projects can be achieved by requiring that some people participate in several, successive Version Projects. This is particularly important for the participants from the computer department staff.

Each Version Project has the main characteristics of a project, but since the projects are repeated, some of the characteristics of a line organisations are also present:

1. A project is characterised in that the task to be performed is unique, while a line organisation also accomplish routine tasks. Each Version Project will design a unique enhancement of the computer based system, while the basic elements of the computer system are known.
2. Another characteristic of a project is that the task to be performed is complex. In a Version Project the tasks might from time to time be more simple than usual in design projects, ie, the tasks can be more similar to "line tasks".
3. A project is a temporary organisational unit, the dissolving is planned already when the project is established. This contradicts the line organisation principles in which organisational survival is a goal by itself. In projects, rules for co-operation, management, division of work etc, have to be established in short time. In version projects, because of the repetitive nature, these cultural aspects can evolve over time, like in a line organisation.

4. A project is goal-oriented in a more concrete, time restricted, and situated way than a permanent organisation. Version Projects also need to relate to the continuity of the line organisation. Project establishment is important for deciding on goals, priorities etc, but does not has to be as time consuming as in other projects.

4.2. Priority workshop as a technique in redesign

Priority Workshops as a technique in redesign is an institutionalised series of workshops throughout the life cycle of the computer system. Priority Workshops can be used as a technique for establishing a Version Project at the beginning of a redesign cycle, in which the objectives, plans, and organisational form are decided. The technique can also be used at particular checkpoints, ie, when major decisions or priorities are to be taken (also within a traditional line organisation). The stages in a Priority Workshop are outlined in Figure 1.

A Priority Workshop aims at (Braa 94):

- putting priorities made by the designers on the agenda,
- making the design priorities transparent,
- creating an possibility for users to influence the design decisions,
- facilitating an overview of different use practices,
- avoiding accidental decisions,
- supporting communication between designers and users.

The basis for making priorities should be the actual use of the system, including the possibly unintended, new ways of using the system that any user group has developed in order to adapt the system to their work. Every workshop therefore must include a discussion on how the computer based system actually is used. By involving users from different user groups in the workshop, the technique aims at creating a collective overview of the different ways of using the actual computer system. This overview is an important part of the knowledge used in making priorities between requests for changes. The consequences of the decisions should be discussed and made explicit to those who may be affected by the changes. Visualisation of possible realistic alternatives may facilitate the discussions of consequences of system changes, in particular if the size and extent of change are substantial. The technique emphasises that communication between developers and users should be established directly and not through mediators.

Participants in the workshop should include developers from different levels of the developer organisation, and users representing different departments and interest groups. It is particularly important that representatives of the employees who actually make requests: the end users participate. The actual decision makers, often represented by a project leader and/or a department manager, should be present, in addition to programmers, who carry out the development work.

Braa (94) describes an experiment of Priority Workshops in a redesign situation, with participants from a development organisation and several of its user organisations. In order to organise decision making from a participatory perspective, the user participants must have a possibility to take part in the design process, though the decision process must be transparent. The experiment shows that Priority Workshop provides such an possibility.

The technique facilitates discussion and evaluation of decision making on an early stage. The transparency should ease the possibilities for the involved parties to intervene the change process. Occasional changes performed because of, eg, power or personal relations to the maintenance staff can then be minimised. Priority Workshops is a technique for users' influence on redesign of a computer based system.

Stage 1. Introductory discussion on the aim of the Priority Workshop.
Stage 2. The users present some good, some bad and some desirable properties of the system in question.
Stage 3. The developers present priorities, plans and ideas concerning the actual system of concern.
Stage 4. Prototypes or mock-ups are presented as examples of design ideas or as a state of the art.
Stage 5. Plenary discussion of the prototypes in light of the presented priorities and desirable properties of the current system.
Stage 6. A summary of the priorities and the properties presented in Stage 3 is made and voted on.
Stage 7. Discussion on which organisational consequences the planned changes or priorities imply for the different user groups.
Stage 8. Summary. The workshop leader sums up the main points discussed during the session, in order to get feedback on disagreements or misunderstandings.

Figure 1. Stages in a Priority Workshop. A more thorough description of Priority Workshops can be found in Braa (94).

5. CONCLUDING REMARKS

In this paper we have discussed the lack of organisation and planning of redesign. We have argued for three main problems of post implementation activities, formulated as dilemmas, with reference to empirical findings and literature studies. The Version Project and Priority Workshop are suggested in order to handle these dilemmas:

1. System development aims at designing stable computer systems for changing environments: The organisation of Version Projects makes it possible to create new designs and still keep the continuity of the current system in mind. The work activities of redesign are organised in an evolutionary organisational form, as a sequence of cycles containing (re)design and use. The line staff is responsible for the continuity of the system, and on this basis makes the evaluation of the needs for a new version. Priority Workshop is a technique for gaining overview over different use contexts, and different, and even contradictory user interests. Priority Workshops can be used in different organisational forms, both linear and project-based.

2. The activities requiring most time and effort are based on the weakest planning and organisation: Version Projects is a way of handling this dilemma because it provides a proposal for structuring the post implementation work. Priority Workshop contribute to make the plans and priorities of change available for discussion and evaluation in an early stage. The involvement of several groups of users, creates a possibility for utilising the differences in knowledge of the various participants.

3. Despite the volume of user initiated changes, user participation is not on the agenda in redesign activities: Version Projects preserve the formal structures for user participation just as in traditional design projects. In order to organise decision making from a participatory perspective, the user participants must have possibility to take part in the design process. Priority Workshop contributes to widen the circle of decision makers and address use quality in redesign activities.

Better planning of redesign may result in increased functional and technical quality in the actual computer systems. As a way of realising a general evolutionary model of system development including redesign, we have suggested to organise redesign in Version

Projects, and to support the decision making with the participatory technique Priority Workshops.

ACKNOWLEDGEMENTS

We would like to thank our colleagues in the FIRE project for inspiring discussions. In addition, the participants in the 15. IRIS provided useful comments. In particular, we thank Bob Galliers and Jacob Nørbjerg for their constructive critique.

REFERENCES

Basili, V. R. (1990) Viewing Maintenance as Reuse-Oriented Software Development, in *IEEE Software*, No. 1, pp 19-25.

Beath, C. M. & Swanson, E. B. (1990) Departementalization in Software Development and Maintenance, in Management of Computing, *Communications of the ACM*, Vol 33 No 6.

Bendifallah, S. & Scacchi, W. (1987) Understanding Software Maintenance Work, in *IEEE Transactions on Software Engineering*, Vol SE-13, no. 3, pp 311-323.

Biskup, H. & Kautz, K. (1990) Maintenance, Nothing else but Evolution?!, *in* Hellman et al (eds) "Precedings of the 13th IRIS", pp 513-528, Åbo Akademi University.

Bjerknes, G., Ehn, P. & Kyng, M. (ed) (1987) "Computers and Democracy: A Scandinavian Challange", Avebury Gower Publishing Company Ltd, Aldershot.

Bjerknes, G., Bratteteig, T. & Espeseth, T. (1991) Evolution of Finished Computer Systems: The dilemma of enhancement, in *Scandinavian Journal of Information Systems*, Vol 3, pp 25-45.

Braa, K. (1992) Influencing System Quality by Using Process Documentation in Prototyping Projects, *in* Muller et al. (eds) "Proceedings of the Participatory Design Conference", Cambridge MA US.

Braa, K. (1994) Priority Workshops as a Springboard for Participatory Design in Redesign Activities, *in* Kerola et al, (eds) "Precedings of the 17th IRIS", pp 875-890, University of Oulo.

Braa et al, (1992): "ENtry to the FIRE project", Department of Informathics, University of Oslo.

Bratteteig, T. & Øgrim, L. (1992) "Strategies for cooperation and development. Report from a preliminary project to the FIRE project", (in Norwegian) Strategier for samarbeid og utvikling. Rapport fra et forprosjekt til FIRE-prosjektet, FIRE report 5, Department of Informatics, Universitety of Oslo.

Briefs, U., Ciborra, C., and Schneider, L. (eds) (1983): "Systems Design For, With, and By the Users" North-Holland, Amsterdam.

Chikofsky, E. J. & Cross, J. H. (1990) Reverse Engineering and Design recovery: A Taxonomy, in *IEEE Software*, Vol January 1990.

Curtis, B., Krasner, H. & Iscoe, N. (1988) A Field Study of the Software Design Process for Large Systems, in *Communications of the ACM*, Vol 31 No. 11.

Directorate of Labour Inspection (1978) "Workers Protection and Working Environment Act" (in Norwegian) Lov om arbeidervern og arbeidsmiljø, 1978.

Espeseth, T. (1992) "Problems and solutions in maintenance of large computer systems" (Master Thesis) (in Norwegian) Problemer og løsninger i forvaltning av store edb-systemer, Department of Informatics, University of Oslo.

Evenshaug, B. (1994): "An Evolutionary Modell for Software Testing" (Master Thesis), (in Norwegian) En evolusjonær modell for programvaretesting, Department of Informatics, University of Oslo.

Floyd, C. (1987) Outline of a Paradigm Change in Software Engineering, *in* Bjerknes et al (eds) "Computers and Democracy", pp 191-210, Avebury Gower Publishing Company Ltd, Aldershot.

Gasser, L. (1986) The Integration of Computing and Routine Work, in *ACM Transactions on Office Information Systems*, Vol 4, No. 3, pp 205-225.

Greenbaum, J. & Kyng, M. (ed) (1991) "Design at Work: Cooperative Design of Computer Systems", Lawrence Erlbaum Ass, New Jersey.

IEEE (1990) "Special Issue on Reverse Engineering".

Hetzel, B. (1988): "The Complete Guide to Software Testing" (third edition) John Wiley & Sons.

Kaasbøll, J. & Øgrim, L. (1994) User Participation in Redesign. A Study of 8 Redesign Processes, *in* Kerola et al, (ed) "Precedings of the 17th IRIS", pp 875-890, University of Oulo.

Lientz, B. P. & Swanson, E. B. (1980) "Software Maintenance Management, A Study of the Maintenance in 487 Data Processing Organisations", Addison-Wesley Publishing Company, Reading, Mass.

Lientz, B. P. & Swanson, E. B. (1981) Problems in application software maintenance, in *Communications of the ACM*, Vol 24 No 11, pp 763-769.

Lientz, B. P. & Swanson, E. B. (1986) "Software Maintenance Management", Addison-Wesley Publishing Company, Reading, Mass.

Lientz, B. P., Swanson, E. B. & Tompkins, G. E. (1978) Characteristics of application software maintenance, in *Communications of the ACM*, Vol 21 No 6, pp 466-471.

LO-NHO (1990) "General Agreement 1990-1993 between Norwegian Federation of Trade Unions and Confoderation of Norwegian Industri (LO-NHO) and Agreement Conserning Computer Based Systems" (in Norwegian) Hovedavtalen 1990-1993 LO-NHO og Rammeavtale vedr. datamaskinbaserte systemer Med kommentarer, Landsorganisasjonen i Norge, Oslo 1990.

Martin, J. & McClure, J. (1983): "Software Maintenance: The Problem and its Solutions" Prentice Hall.

Myers, G.J. (1979): "The Art of Software Testing" John Wiley & Sons.

Namioka & Schuler (ed) (1992) "Perspectives on Systems Design: Participatory Design", Lawrence Erlbaum Ass, New Jersey, US.

Naur, P. (1985) Programming as Theory Building, in *Microprocessing and Microprogramming*, Vol 15, pp 253-261, North-Holland Publishing Company.

Swanson, E. B. (1976) The dimensions of maintenance, in "Proceedings 2nd International Conference on Software Engineering", 13-15 October, IEEE Computer Society, pp 482-497, San Fransisco, California, Long Beach, CA.

Sørgaard, P. (1989) "An Overview of Research in Maintenance", Report from Åbo akademi, informationsbehandling & matematik Ser. A, No 94, Åbo akademi, Departments of Computer Science and Mathematics.

Øgrim, L. (1992) "Maintenance or redesign, a question of organising?" (in Norwegian) Vedlikehold eller redesign, et spørsmål om organisering? FIRE report no 3, Department of Informatics, University of Oslo.

OBJECT AND PROGRAM REUSABILITY MECHANISMS

Xavier Castellani and Hong Jiang

CEDRIC-IIE (CNAM) research laboratory, 18 allée Jean Rostand, 91025 Evry Cedex, France

Abstract

This paper describes standardized reusability mechanisms of object characteristics using a metric and a guideline. These mechanisms find seven results of object reusability which allow us to merge objects, to define inheritances between objects, to create abstract objects and to reuse characteristics of objects.

This paper presents an application of these object reusability mechanisms on program code. These mechanisms find five results of code reusability which allow us to merge programs, to store programs in a library and to include them in other programs, and to store common sequences code of programs in a library and to include them in programs.

These mechanisms belong to MCO object-oriented analysis and design methodology. They are presented with concepts and the vocabulary of MCO but they are general. They can be used with concepts of other object-oriented analysis and design methods, languages and data base management systems.

1. INTRODUCTION

According to Nauer and Randell [19], *software reuse is the process of creating software systems from existing software rather than building software systems from scratch.*

According to Meyer [18], *reusability is the ability of software products to be reused, in whole or in part, for new applications.*

The software reuse in developing new software is a well known field in software engineering for about twenty years [16]. There is an increasing need for a corresponding maturity in reutilisability, assessment and measurement. The assessment of a component's quality and reutilisability are critical to a successful reuse effort. Components must be easily comprehensible, easily incorporated into new systems. Unfortunately, no consensus currently exists on how to go about measuring component's reutilisability [4].

The success of reuse technology depends on the integration of reuse libraries into the design and programming environments. To reuse software, it is necessary to have specifications of the software. Without these specifications the reuse is impossible. Furthermore, it is very inefficient to look through manually the specifications [9]. When the number of reusable components is quite big, this is not practical. Therefore, reutilisability mechanisms and on-line searching mechanisms are required. With the support of an on-line searching tool [3], software can be shared easily.

This paper presents a solution to normalize object and program reusability with mechanisms which use metrics and guidelines. These mechanisms may be used on specifications of object characteristics and on program code.

The reusability mechanisms allow the study of object similarity. Similarity is a general concept which determines a comparison between any entities [20] [24]. Works on object similarity use criteria and rules as the TELOS project [22].

The idea of using metrics to normalize the object reusability has been proposed by several object-oriented specialists. We can mention Cox [11], Chidamber and Kemerer [10], Griss who has indicated in [12] that metrics would be used in the "Corporate Engineering Software Reuse HP' program", and groups of the Annual Workshop on Software Reuse [25]. To our knowledge, it does not exist rational utilization of metrics to normalize the reusability of objects or programs.

The object reusability mechanisms allow the creation of object merges, simple inheritances and abstract objects. The principle of the mechanisms had been presented in [8]. This paper recalls, and specifies them in Section 2.

The application of these mechanisms on programs allow us to find common programs and common sequences in different programs to store them in libraries and to include in programs. The program code standardized reusability mechanisms are presented in Section 3.

2. OBJECT STANDARDIZED REUSABILITY

Only the concepts of the MCO model which allow us to understand the object reusability mechanisms are presented in Section 2.1. The seven results of object reusability are presented in Section 2.2. The object reusability mechanisms are presented in Section 2.3.

2.1 Concepts of the MCO Model which allow the Understanding of the Object Reusability Mechanisms

MCO is an object-oriented analysis and design methodology. The goal of this paper is not to present MCO [6] [7] and we underline that the reusability mechanisms may be used by other methods.

- *Inheritance*

An object type OB-b inherits from an object type OB-a if the set of the non masked characteristics of OB-a is included in the set of the characteristics of OB-b.

An object type OB-b inherits from an object type OB-a with an inheritance by specialization if the instances of OB-b are elements of the set of the instances of OB-a. This definition is conform with the definition of the specialization-generalization links "Is a" of Smith and Smith [21].

An object type OB-b inherits from an object type OB-a with a construction inheritance if the instances of OB-b are not elements of the set of the instances of OB-a. The construction inheritances have not semantics; they are only defined to reuse characteristics.

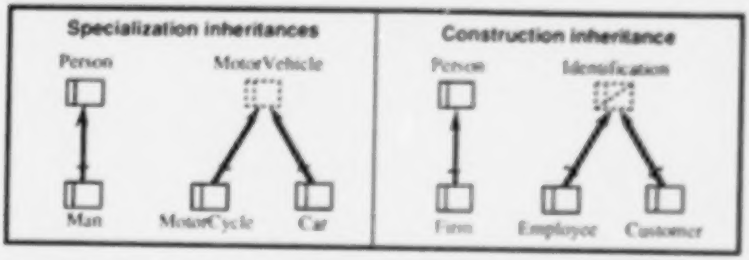


Figure 1: Examples of specialization inheritances and construction inheritance

- ***Abstract object.*** *An abstract object is an object for which no instance can be created.* Example: the object MotorVehicle is abstract if the instances of vehicle are only the instances of Car, Motorcycle, etc., which inherit from it.

- ***Reusability of object characteristics by declaration***

The reusability of object characteristics by declaration allows the reuse of object characteristics (the types of the attributes, the services), considered one by one, directly, independently from inheritances. The reusability by declaration is used with languages such as Ada, Pascal, etc.

2.2 The Seven Results of Object Reusability

The seven possible results of object reusability between two objects types OB-a and OB-b are in Figure 2.

1 - OB-b is of the type of OB-a: OB-a and OB-b are identical.
2 - OB-b is merged with OB-a.
3 - OB-b inherits from OB-a.
4 - OB-a inherits from OB-b.

5 - Creation of an abstract object from which OB-a and OB-b inherit.

6 - Characteristics of OB-b are declared to be of the types of characteristics of OB-a.
7 - Characteristics of OB-a are declared to be of the types of characteristics of OB-b.

Figure 2: The seven possible results of reusability between two objects types

2.3 The Object Reusability Mechanisms

After presentation of the object reusability guideline in Section 2.3.1, reusability mechanisms which use this guideline are presented in Section 2.3.2.

2.3.1 The Object Reusability Guideline

The object reusability guideline allows the creation of object merges, simple inheritances and abstract objects from which other objects inherit. *It fixes:*

. the weight of the minimum of common characteristics of two objects (WeightMinCCOB),

. and the weight of the maximum of particular characteristics of an object with regard to an other object (WeightMaxPCOB),

in order that these two objects may be merged, or one may inherit from the other, or an abstract object may be created from which these two objects inherit.

The common characteristics of two objects are the attributes which are of the same type and the services which provide the same processing.

Two services provide the same processing if they have the same signature and if their specifications are syntactically identical or if a tool or an analyst must certify that they ensure the same processing.

The particular characteristics of two objects are their characteristics which are not common to these objects. If an object OB-b inherits from an object OB-a, the particular characteristics of OB-a must be masked in OB-b. If the masking is forbidden, it is impossible. It means that the measure, the weight, of these particular characteristics must be equal to zero.

2.3.2 Presentation of the Object Reusability Mechanisms

The reusability mechanisms of objects optimize the reusability of their characteristics with the following criteria:

. a characteristic used by several objects must be defined in only one of these objects,

. the reusability must be made in priority with groups of characteristics (by re-using objects) if their characteristics satisfy the reusability guideline constraints, otherwise by defining individual reusability by declaration of the characteristics,

. and the number of objects must be minimized.

Let us consider a new object type OB-b to be created and an existing object type OB-a in an object system which has the most weight of characteristics in common with OB-b.

The weight of their common characteristics, Weight (CCOB-ab), may be superior or equal to the minimum fixed by the reusability guideline, WeightMinCCOB, to create a merge of these two objects, a simple inheritance between these two objects or an abstract object from which these two objects inherit.

The weight of particular characteristics of OB-a compared with OB-b, Weight (PCOB-a), and the weight of particular characteristics of OB-b compared with OB-a, Weight (PCOB-b), are compared with the maximum weight of particular characteristics fixed by the reusability guideline, WeightMaxPCOB.

The mechanisms of reusability of a new object OB-b and an existing object OB-a which has the most weight of common characteristics with OB-b are defined by the decision table presented in Figure 3. The seven results of this decision table are those presented in Section 2.2.

- The characteristics of OB-b are the same that the characteristics of OB-a.	Yes	No					
- Weight of common characteristics of OB-a and of OB-b: **Weight (CCOB-ab)**, compared with the minimum weight of common characteristics fixed by the object reusability guideline ***WeightMinCCOB***.	/	≥				<	
- Weight of particular characteristics of OB-a: **Weight (PCOB-a)**, compared with the maximum weight of particular characteristics fixed by the object reusability guideline ***WeightMaxPCOB***.	/	≤		>		/	/
- Weight of particular characteristics of OB-b: **Weight (PCOB-b)**, compared with the maximum weight of particular characteristics fixed by the object reusability guideline ***WeightMaxPCOB***.	/	≤	>	≤	>	/	/
- The reusability by declaration is allowed.	/	/	/	/	/	Yes	No
1 - OB-b is of the type of OB-a: OB-a and OB-b are identical.	X						
2 - OB-b is merged with OB-a.		X					
3 - OB-b inherits from OB-a.			X				
4 - OB-a inherits from OB-b.				X			
5 - Creation of an abstract object from which OB-a and OB-b inherit.					.X		
6 - Characteristics of OB-b are declared to be of the types of characteristics of OB-a. **7 - or characteristics of OB-a are declared to be of the types of characteristics of OB-b.**						X	
- OB-b does not reuse OB-a and OB-b is not reused by OB-a.							X

Figure 3: Decision table to study the reusability between two objects types

3. PROGRAM CODE STANDARDIZED REUSABILITY

In this Section we apply to program and sequence code reusability the principle of the object reusability mechanisms. The seven results of object reusability bring along five results presented in Section 3.1. Metrics to measure program code are presented in Section 3.2. The program code reusability mechanisms are presented in Section 3.3. An example of program code standardized reusability is given in Section 3.4.

- Common Sequences Code and Particular Sequences Code of Two Programs

A sequence code is a set of lines of code. We distinguish *declaration sequences (DS) which are data declarations and the other declarations: declarations of macros, of "include", of "copy", etc. which can be used, from processing sequences (PS) which can be performed.*

BEST COPY AVAILABLE

The common declaration sequences of two programs are the data declarations of the same types and the other declarations which are identical: declarations of macros, declarations of "include" or of "copy", etc.

Data declarations of the same types may be declarations of data which have not the same names. Common declarations sequences may be not contiguous sequences and may be not ordered with the same order. Numerous programs have common sequences. It is in particular the case for input/output programs.

The particular declaration sequences of two programs are the declarations which are not common declaration sequences.

The common processing sequences of two programs are processing sequences which provide the same processing.

Two processing sequences provide the same processing if their specifications are syntactically identical or if a tool or an analyst must certify that they ensure the same processing. It is in particular necessary to study processing sequences defined with data differently named of the same types, and to study processing sequences not specified with the same statements, for example "case" statements specified with "if" statements. The common processing sequences may be not contiguous sequences. For example in Cobol, a processing sequence may be defined with statements using "Perform" blocks and these blocks. Common processing sequences may be not ordered with the same order. For example "Allocation" statements, blocks of the different cases of "case" statements and parallel blocks of statements.

The particular processing sequences of two programs are the processing sequences which are not common processing sequences.

3.1 The Five Results of Program Code Reusability

The five possible results of program code reusability between two programs P-a and P-b are deduced from the seven results of object reusability presented in Section 2.2.

1 - P-b is identical to P-a.
2 - P-b is merged with P-a.
3 - P-a is stored in a library and is "included" in P-b.
4 - P-b is stored in a library and is "included" in P-a

5 - One or several common sequences existing in both programs P-a and P-b are stored in a library and are "included" in P-a and in P-b.

Figure 4: The five possible results of reusability between two programs

3.2 Metrics to Measure Program Code

Numerous software experts have proposed metrics to measure software components. Some of them are mentioned by Fenton [13]. The simplest unit to measure source program code length is Lines of Code (LOC). Others prefer the Used Instructions (UI): the COCOMO method (COst COnstructive MOdel) [5] of Boehm for example. Others use the Function Points (FPs) [2]. However, most companies use LOC in spite of their deficiencies [14]. Software engineering specialists have defined other metrics than the sizes of the programs to measure the quality of the software. Among the most used the metrics to measure numbers of operators and numbers of operands, mention the metrics of Halstead [15], the metrics to measure program control graphs, mention the metrics of McCabe [17] and the metrics to measure program call graphs, the hierarchical complexity which is the average number of modules by level. In this article we use the sizes to measure programs code and we measure these sizes with lines.

3.3 The Program Code Reusability Mechanisms

3.3.1 The Program Code Reusability Guideline

The code reusability guideline is similar to the object reusability guideline. *For two programs, it fixes:*

. the minimum size of their common sequences (MinCL),

. and the maximum size of the particular sequences of a program with regard to the other (MaxPL),

in order that these two programs may be merged, or one may be stored in a library and "included" in the other.

For two identical sequences of code, MinCL fixes their minimum size in order that one of them may be stored in a library and "included" in their programs.

- Which code reusability guideline should we choose?

It is not difficult to define a code reusability guideline in an information department because analysts and developers have precise ideas of their values. Generally, the average size of a written in C, Pascal, Cobol, ..., is comprised between 800 and 3000 lines; some programs even pass beyond 5000 lines. With these languages, the sequences which are stored in libraries to be reused in several programs with the statements "Include" in C or Pascal, "Copy" in Cobol, ..., have generally minimal sizes comprise between 30 and 100 lines according to the programming recommendations: "it is forbidden to store too little sequences in a library". These minimal sizes (30 and 100 lines), are possible values of MinCL. MinCL must be superior to one line. With non object-oriented programming languages the masking is not usable, hence MaxPL must be count in tens lines (and the particular sequences are defined in case sequences).

3.3.2 Presentation of the Program Code Reusability Mechanisms

This criteria used by the program code reusability mechanisms are an adaptation of the criteria of the object reusability mechanisms presented in Section 2.3.2.

The code reusability mechanisms optimize the reusability of program code with the following criteria:

. a similar program code reused in several programs and satisfying the reusability guideline constraints must be defined once in a library,

. a sequence code reused several times in programs and satisfying the reusability guideline constraints must be defined once in a library,

. the reusability must be made in priority with programs otherwise with sequences,

. and the number of programs and of sequences reused must be minimized.

The mechanisms of reusability of a new program P-b with an existing program P-a which has the most lines in common with P-b, are defined by the decision table presented in Figure 5. The five results of this decision table are presented in Section 3.1.

The code of P-a and the code of P-b are identical	**Yes**	**No**					
Number of lines of common sequences of P-a and P-b **Number(CL-ab)**, compared with the minimum of number of lines fixed by the reusability guideline ***MinCL***.	/	≥					<
Number of lines of the particular sequences of P-a **Number(PL-a)**, compared with the maximum number of particular lines fixed by the reusability guideline ***MaxPL***.	/	≤		>			
Number of lines of the particular sequences of P-b **Number(PL-b)**, compared with the maximum number of particular lines fixed by the reusability guideline ***MaxPL***.	/	≤	>	≤	>		
Number of lines of sequences of **data declarations** (DS-i and DS-j) or of processing statements (PS-i and PS-j) of P-a and P-b **Number(CL-ij)**, compared with the minimum of number of lines fixed by the reusability guideline ***MinCL***.					≥	<	
1 - P-b is identical to P-a.	**X**						
2 - P-b is merged with P-a.		**X**					
3 - P-a is stored in a library and is "included" in P-b.			**X**				
4 - P-b is stored in a library and is "included" in P-a.				**X**			
5 - One or several common sequences existing in P-a and P-b are stored in a library and are "included" in P-a and in P-b.					**X**		
No reusability between P-a and P-b						**X**	**X**
	Rule CR1	**Rule CR2**	**Rule CR3**	**Rule CR4**	**Rule CR5**	**Rule CR6**	**Rule CR7**

Figure 5: Decision table to study the reusability between two programs P-a and P-b

3.4 Example of Program Code Standardized Reusability

Two C functions presented in [23] are in Figure 6:

. a function to be created: "plots" which converts from system coordinates to EGA high resolution screen coordinates and plots a point on the screen in the designated color;

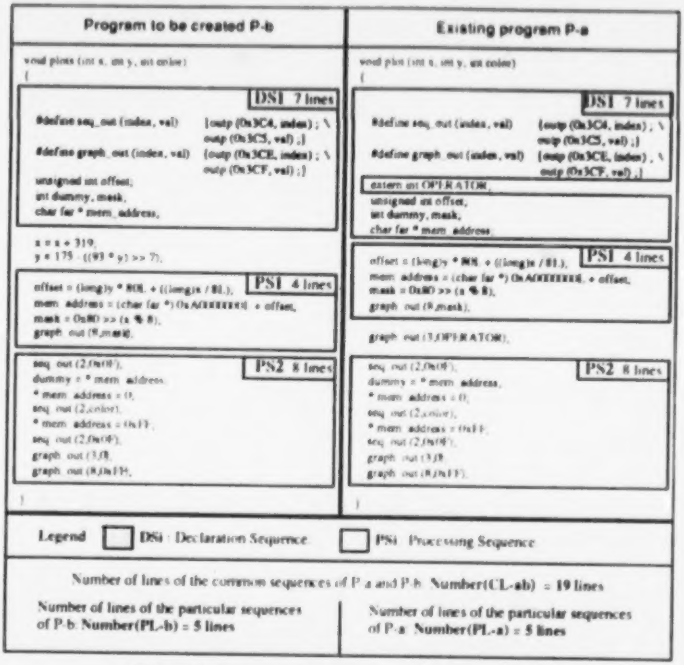
. and an existing function: "plot" which plots a point at (x,y) in color for Enhanced Graphics Adapter, using Turbo C port output functions.

"plot" is the existing function which has the most lines in common with "plots". It is similar to the previous function. It is a particular case but these functions are short and can be wholly presented in this paper. The common declaration sequences of these functions and their common processing sequences are contiguous except DS1, and are ordered with the same orders. We study the reusability of these two functions on considering the lines of code.

The results of the program reusability mechanisms are presented in Figure 7 according to several values of the code reusability guideline. The links "is included in" are represented with the arrows which represent the object inheritances construction.

4. CONCLUSION

The reusability guidelines allow comparison of objects or programs to study if they are similar by measuring respectively their common characteristics or their code and their particular characteristics and their particular sequences. The object reusability



Program to be created P-b

```
void plots (int x, int y, int color)
{
#define seq_out (index, val)      {outp (0x3C4, index) ; \
                                   outp (0x3C5, val) ;}
#define graph_out (index, val)    {outp (0x3CE, index) ; \
                                   outp (0x3CF, val) ;}
unsigned int offset;
int dummy, mask;
char far * mem_address;
x = x + 319;
y = 175 - ((93 * y) >> 7);
offset = (long)y * 80L + ((long)x / 8L);
mem_address = (char far *) 0xA0000000L + offset;
mask = 0x80 >> (x % 8);
graph_out (8,mask);
seq_out (2,0x0F);
dummy = * mem_address;
* mem_address = 0;
seq_out (2,color);
* mem_address = 0xFF;
seq_out (2,0x0F);
graph_out (3,0);
graph_out (8,0xFF);
}
```

Existing program P-a

```
void plot (int x, int y, int color)
{
#define seq_out (index, val)      {outp (0x3C4, index) ; \
                                   outp (0x3C5, val) ;}
#define graph_out (index, val)    {outp (0x3CE, index) ; \
                                   outp (0x3CF, val) ;}
extern int OPERATOR;
unsigned int offset;
int dummy, mask;
char far * mem_address;
offset = (long)y * 80L + ((long)x / 8L);
mem_address = (char far *) 0xA0000000L + offset;
mask = 0x80 >> (x % 8);
graph_out (8,mask);
graph_out (3,OPERATOR);
seq_out (2,0x0F);
dummy = * mem_address;
* mem_address = 0;
seq_out (2,color);
* mem_address = 0xFF;
seq_out (2,0x0F);
graph_out (3,0);
graph_out (8,0xFF);
}
```

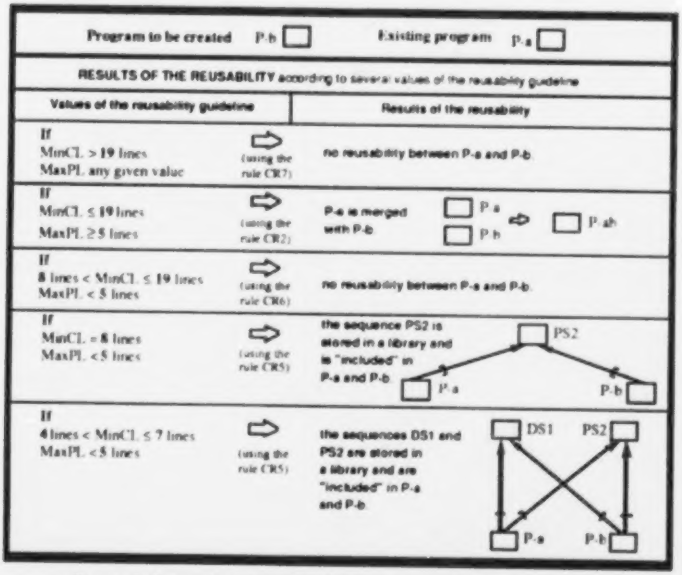
Legend ☐ DSi : Declaration Sequence ☐ PSi : Processing Sequence

Number of lines of the common sequences of P-a and P-b: **Number(CL-ab) = 19 lines**

Number of lines of the particular sequences of P-b: **Number(PL-b) = 5 lines**

Number of lines of the particular sequences of P-a: **Number(PL-a) = 5 lines**

Figure 6: Example of C program reusability study



Program to be created P-b ☐		Existing program P-a ☐
RESULTS OF THE REUSABILITY according to several values of the reusability guideline		
Values of the reusability guideline		**Results of the reusability**
If MinCL > 19 lines MaxPL any given value	⇨ (using the rule CR7)	no reusability between P-a and P-b
If MinCL ≤ 19 lines MaxPL ≥ 5 lines	⇨ (using the rule CR2)	P-a is merged with P-b
If 8 lines < MinCL ≤ 19 lines MaxPL < 5 lines	⇨ (using the rule CR6)	no reusability between P-a and P-b
If MinCL = 8 lines MaxPL < 5 lines	⇨ (using the rule CR5)	the sequence PS2 is stored in a library and is "included" in P-a and P-b
If 4 lines < MinCL ≤ 7 lines MaxPL < 5 lines	⇨ (using the rule CR5)	the sequences DS1 and PS2 are stored in a library and are "included" in P-a and P-b

Figure 7: Examples of results of program standardized reusability (obtained with several values of the reusability guideline)

BEST COPY AVAILABLE

mechanisms have been implemented on Smalltalk-80 [3]. They should be implemented in CASE tools, in DBMS and in programming environments of languages.

An outstanding research is on the possibility to take into account programs and sequences code syntactically different but which have nearly the same control graph and/or the same call graph.

5. REFERENCES

[1] J.R. Abrial, Data semantics in data management, North Holland, 1974.

[2] A.J. Albrecht, Measuring Application Development Effort Productivity, Proceedings Joint IBM/SHARE/GUIDE App. Symp., October 1979.

[3] G. Blaise, Implantation de l'algorithme de réutilisabilité uniforme d'objets MCO en Smalltalk-80. Graduate thesis engineer cycle C CNAM, 1992.

[4] G. Boetticher, K. Srinivas, D. Eichmann, A neural net-based approach to software metrics. The fifth international conference on software engineering and knowledge engineering, 1993.

[5] B.W. Boehm, Software Engineering Economics, Prentice-Hall, New York, 1981.

[6] X. Castellani, Le modèle de la méthode MCO d'analyse et de conception des systèmes d'objets, INFORSID Conference, Paris, June 5-7 1991.

[7] X. Castellani, MCO: Méthodologie générale d'analyse et de conception des systèmes d'objets, Tome 1: L'ingénierie des besoins, preface of C. Rolland, 1993, Masson.

[8] X. Castellani, Mechanisms of Standardized Reusability of Objects, IFIP WG 8.1 Working Conference on Information System Development Process Como, September 1-3, 1993, proceedings published by North-Holland.

[9] J. Cheng, Improving the software reusability in object-oriented programming, ACM Press, Oct 1993.

[10] R. Chidamber and F. Kemerer, Towards a metrics suite for object oriented design, OOPSLA '91, Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM SIGPLAN NOTICES, Volume 26, Number 11, November 1991.

[11] B. Cox, Planning the software industrial revolution, IEEE Software, 7(6), November 1990.

[12] P.M. Cullough, B. Atkinson, A. Golberg, M. Griss and J. Morrinson, Reuse: Truth or Fiction (PANEL), OOPSLA '92, Seventh annual Conference on Object-Oriented Programming Systems, Languages, and Applications, ACM SIGPLAN NOTICES, Volume 27, Number 10, October 1992.

[13] N.E. Fenton, Software metrics, A Rigorous Approach, Chapman & Hall, 1991.

[14] D.G. Firesmith, Managing Ada projects: the people issues, Proceedings TRI Ada'88, Charleston, WV, 24-27 October 1988.

[15] M.H. Halstead, Element of Software Science, Elsevier, North Holland, 1975.

[16] C. Krueger, Software reuse, ACM Computing Surveys, 24(2), 1992.

[17] T.J. McCabe, A complexity measure, IEEE Transactions on Software Engineering, SE-2(4), December 1976.

[18] B. Meyer, Object-oriented Software Construction. Prentice Hall international, Series in computer science, 1988.

[19] P. Nauer and B. Randell, Software Engineering: Report on a Conference by the NATO Science Committee. NATO Scientific Affairs Division, Brussels, Belgium, 1968.

[20] L. Sjoberg, A Cognitive Theory of Similarity, Goteborg Psychological Reports, Number 10, Volume 2, 1972.

[21] J.M. Smith and D.C.P. Smith, Database Abstractions: Aggregation and Generalization. ACM Transactions on Database Systems, vol 2, n 2, June 1977.

[22] G. Spanoudakis and P. Constantopoulos, Similarity for Analogical Software Reuse: A Conceptual Modeling Approach, 5th International Conference CAISE '93, Paris, June 1993, Springer-Verlag.

[23] R.T. Stevens, Graphics Programming in C, 1990, M&T Publishing, Inc.

[24] A. Tversky, Features of Similarity, Psychological Review, 44(4), July 1977.

[25] WISR'92: 5th Annual Workshop on Software Reuse Working Group Reports, M. Griss and Will Tracz editors, Software Engineering Notes, ACM Press, Volume 18, Number 2, April 1993.

CASE tools

INTEGRATION ISSUES OF INFORMATION ENGINEERING BASED I-CASE TOOLS

Karl Kurbel, Thomas Schnieder

University of Muenster, Institute of Business Informatics, Grevener Strasse 91,
D-48159 Muenster, Germany

Abstract

Problems and requirements regarding integration of methods and tools across phases of the software-development life cycle are discussed. Information engineering (IE) methodology and I-CASE (integrated CASE) tools supporting IE claim to have an integrated view across major stages of enterprise-wide information-system development: information strategy planning, business area analysis, system design, and construction. In the main part of this paper, two comprehensive I-CASE tools, ADW (Application Development Workbench) and IEF (Information Engineering Facility), are analyzed and compared with regard to integration issues.

1. THE "I" OF I-CASE

Many attempts have been made to overcome the so-called "software crisis". One of them is computer-aided software engineering (CASE). CASE tools have been developed in large numbers, but they share a common deficiency: lack of integration. Tool support is often limited just to one phase or a few phases of the development cycle, or just to one view of the problem (e.g. process modelling, but not data modelling) (Nomina, 1993). When isolated tools have to be used, work can become rather cumbersome. Tool integration therefore is an important issue. Lack of integration is a major criticism regarding CASE tools (Stobart et al., 1993, p. 84) because it is detrimental to acceptance of CASE by the users (Zarrella, 1990, p. 208).

Integrated tools do not only enhance development productivity but also help to avoid unnecessary inconsistencies among different representations of the same things. Various approaches aiming at tool integration have been proposed (e.g. Dineur, 1990; Venable, 1990; Wybolt, 1991). The best-known example is probably IBM's Repository Manager. Its underlying idea was a common information model for all tools. IBM's failure after ten years of development effort shows very clearly that tool integration is a difficult task (Polilli, 1992; Bucken, 1992).

Another aspect of integration is related to the fact that real-world IS are rarely stand-alone systems. Instead, they are part of enterprise-wide information management and have to match with other IS. Software engineering, however, takes a rather limited view: It looks at development of just *one* information system at a time, but it does not take into account explicitly that *many* interlocking IS are needed to solve business problems.

A more comprehensive view underlies the information engineering (IE) methodology introduced by James Martin (Martin, 1989). IE proceeds in a top-down manner, starting with information strategy planning (ISP) and continuing with analysis of major sectors of the enterprise (business area analysis, BAA), design of individual IS within this framework (system design, SD), and finally construction of these systems.

Major concerns of IE are integration across stages and tool support. This means that not only techniques employed in the four stages have to match but also that there have to be tools supporting these techniques, and that these tools have to be *I-CASE* (integrated CASE) tools. Two well-known tool sets supporting the comprehensive and rather sophisticated IE philosophy are: ADW (Application Development Workbench) by KnowledgeWare and IEF (Information Engineering Facility) by Texas Instruments Information Engineering.

2. INTEGRATION IN LIFE-CYCLE CASE

Considering the number of papers and software products addressing "integration", this term has become something like a buzzword. In this paper, we discuss integration from two points of view. One is the tool view, i.e. horizontal and vertical integration among tools. The other one is the interpersonal view, related to cooperative development.

Horizontal integration refers to tools employed within one IE stage. There is quite a number of tools available for each stage. They deal with different aspects of the problem (e.g. process decomposition, data flows), or they show the same things in different views. For example, entities appear in entity-relationship (ER) diagrams, in data-flow diagrams, in association matrices, etc. Horizontal integration means that tools can easily exchange their results and that modifications or new results established by one tool are immediately available for other tools. This is particularly important with regard to information interchange between the data view and the activities view of IS development. A major concern of IE is that data modelling and process modelling should go hand in hand. This can only work, however, if the tools supporting either view are truly integrated.

Vertical integration goes across IE stages, or, in general, across life-cycle phases. Results established with tools of one stage should be available for tools of other stages in a natural way. In particular, results should be represented not only as documentation but also in a format that they can be processed automatically by other tools. Two specific aspects are forward and reverse integration. *Forward integration* capability calls for features permitting model objects to be specified on a high abstraction level first and to be expanded and enriched by additional details later. *Reverse integration* capability means that changes made in the representation format of a later stage will be adopted in the models and representation formats of earlier stages. As an example, consider relational database design during SD. If relations with additional entity types are introduced here, those entity types should be represented automatically in the ER models of business area analysis.

Interpersonal Integration: When several people form a development team, results are produced at different places. Distributed development is typical for most real-world IS projects. In this case, distributed results have to be coordinated. The most straightforward solution is a central *repository* all distributed workplaces are connected to and all tools store their results in, but it is difficult to realize. IBM's failure with this approach has already been mentioned. If central repositories are available, they are not free of drawbacks (Kramer, 1991, p. 501); e.g. they often suffer from performance problems. Distributed repositories may reduce some of the problems, but ensuring consistency is even harder here. Techniques to coordinate work results were proposed by Garbajosa et al. (1990) and Robinson (1991), for example. Such techniques are state of the art in CSCW (computer-supported cooperative work), but advanced CSCW concepts have not been implemented in I-CASE tools yet.

3. INTEGRATION ASPECTS OF ADW

ADW (Application Development Workbench) is a large IE tool set distributed by KnowledgeWare Inc. ADW is the successor of IEW (Information Engineering Workbench). It runs on PCs under the OS/2 operating system. Target environments are primarily IBM mainframes under MVS. Since 1992, OS/2-based PCs with Presentation Manager (PM) interface are also supported. Very recently, in version 2.7, MS-Windows has been added to the list of target operating systems. This report and evaluation are mostly based on experience with ADW, version 1.6.04, for OS/2 PM both as development and target environment.

3.1 Tools of ADW

All information collected and produced during IE stages is stored in an *encyclopedia*. Objects of an encyclopedia are, for example, business functions, processes, organizational units, entity types, etc. Developers describe objects by associating so-called *details* with them, and they specify interrelations between the objects.

For this purpose, a set of tools is available. ADW tools are grouped, according to IE stages, into four subsets called workstations (KnowledgeWare, 1994): Planning Workstation, Analysis Workstation, Design Workstation, and Construction Workstation. Figure 1 shows 11 of the tools that were used during the first three stages of one of our projects. There are more tools in the tool set.

The *Association Matrix Diagrammer* is used to specify relations between different types of encyclopedia objects (e.g. which entities are read or written by which processes). Hierarchical relations are described and plotted with the help of the *Decomposition Diagrammer* (e.g. process hierarchies, organizational structures). The *Property Matrix Diagrammer* is applied to specify details of objects (e.g. entity descriptions). The *Entity Relationship Diagrammer* supports data modelling. Connections between data and processes are described by data-flow diagrams created with the help of the *Data Flow Diagrammer*. Coarse procedural logic of elementary processes may be outlined within the *Minispec Action Diagrammer* and later refined by means of the *Module Action Diagrammer*.

3.2 Horizontal Integration

ADW's encyclopedia is a *central* encyclopedia with respect to the tool view, but not with respect to interpersonal cooperation. "Central" means that all pieces of information, no matter which tool they were created with, are stored just once. All tools have access to the encyclopedia. When they read information, it is transformed into a tool-specific format (usually a graphical one) and displayed to the user. When they write information, it is represented in the encyclopedia in a standard format that can be interpreted by other tools. In this way, consistency among tools is no problem.

Information exchange between tools comprises not only object definitions but also relations between encyclopedia objects. For example, hierarchical relations among organizational units, specified by the Decomposition Diagrammer, are also available for the Association Matrix Diagrammer; the latter one creates the relationship "organizational unit *coordinates* organizational unit" from information about the hierarchy.

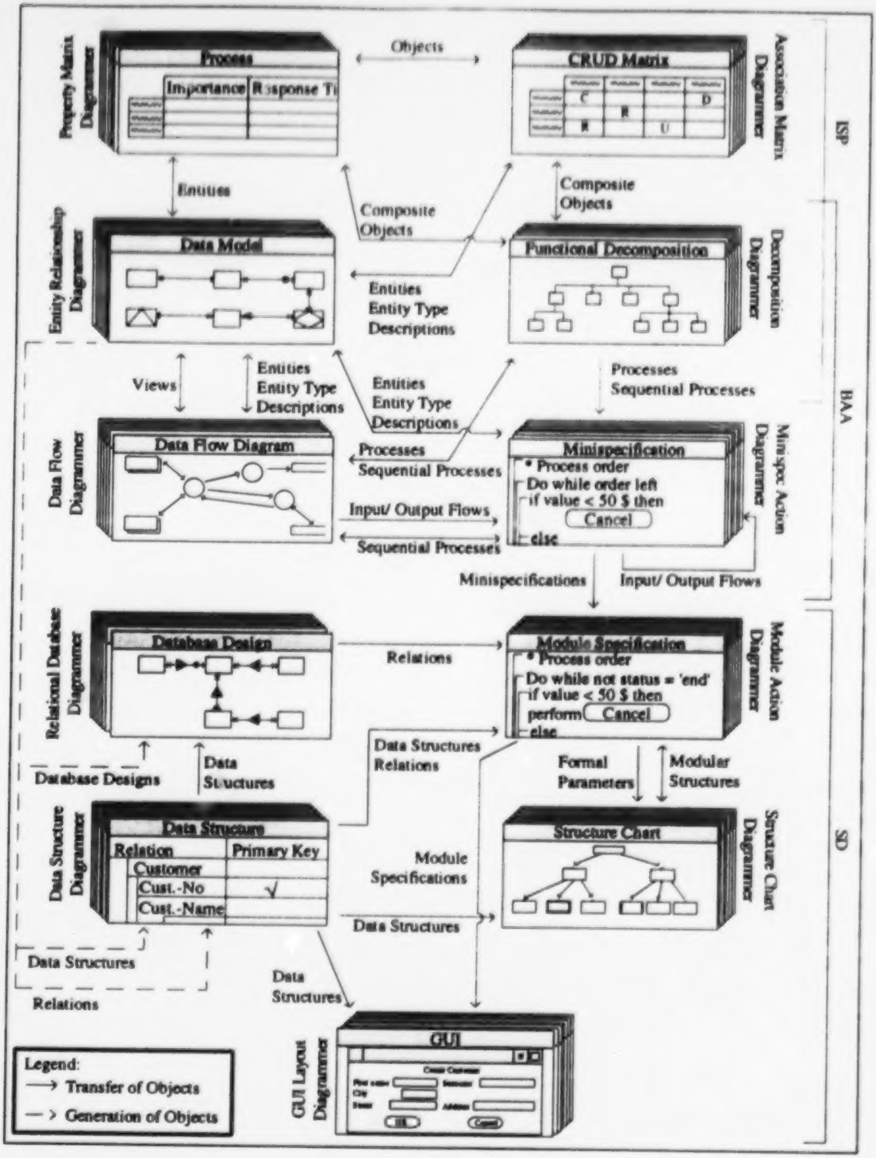

Property Matrix Diagrammer
Process
Importance
Response Ti
Objects
CRUD Matrix
C
D
R
R
U
Association Matrix Diagrammer
ISP
Entities
Composite Objects
Composite Objects
Entity Relationship Diagrammer
Data Model
Functional Decomposition
Decomposition Diagrammer
Entities
Entity Type
Descriptions
BAA
Views
Entities
Entity Type
Descriptions
Entities
Entity Type
Descriptions
Processes
Sequential Processes
Data Flow Diagrammer
Data Flow Diagram
Processes
Sequential Processes
Minispecification
* Process order
Do while order left
if value < 50 $ then
Cancel
else
Minispec Action Diagrammer
Input/ Output Flows
Sequential Processes
Minispecifications
Input/ Output Flows
Relational Database Diagrammer
Database Design
Relations
Module Specification
* Process order
Do while not status = 'end'
if value < 50 $ then
perform
Cancel
else
Module Action Diagrammer
SD
Database Designs
Data Structures
Data Structures
Relations
Formal Parameters
Modular Structures
Data Structure Diagrammer
Data Structure
Relation
Primary Key
Customer
Cust.-No
√
Cust.-Name
Module Specifications
Structure Chart
Structure Chart Diagrammer
Data Structures
Data Structures
Relations
Data Structures
GUI Layout Diagrammer
GUI
Legend:
⟶ Transfer of Objects
– > Generation of Objects

Figure 1. Horizontal and vertical integration in ADW

An advantage of integrated tools is that developers may choose the most appropriate tools themselves if certain results can be achieved by more than one tool. Definition of data flows is an example. The Data Flow Diagrammer provides a formal language to describe data flows, but there is a more user-friendly way, by means of the Entity Relationship

BEST COPY AVAILABLE

Diagrammer. When a view of the data flow is specified there, ADW will derive the formal description automatically, without bothering the user again.

Horizontal integration includes *context-sensitive transition* from one tool to another. This means that a tool may be called from within another one, and that the current context of the first tool will be considered by the second one. If an object has been selected, for example, and a second tool is called afterwards, the latter one will focus on the selected object and display information concerning that particular object and its surroundings. For example, when a process of a data-flow diagram is selected, within the Data Flow Diagrammer, and the Minispec Action Diagrammer is then called, the procedural description of just that process will be displayed. Furthermore, model consistency is supported by context-sensitive transition because informations can be derived automatically instead of entering them by hand. Thus sources of errors are reduced.

3.3 Vertical Integration

Integration across stages is less complete than horizontal integration. In principle, models and diagrams of earlier stages are available in later stages, but there are several restrictions and drawbacks.

Decomposition diagrams (e.g. functional hierarchies) and data models from ISP can be processed by Analysis Workstation tools without any problems. As to *forward integration*, however, a distinction has to be made between hierarchical objects and data. Decomposition diagrams may be further refined by additional levels. For example, business functions can be split up into more detailed functions; those functions can be described by processes which can be further refined by other processes and finally by so-called "elementary processes".

Refining the ISP data model is less satisfactory. For this purpose, features for clustering entities and relationships would be necessary (Teorey, 1989; Rauh, 1992; Jaeschke, to appear), but they are not available. Instead, coarse entities and relationships may be *substituted* by more detailed ones. This means, however, that the ISP model providing a high-level overview is destroyed. Furthermore, entries of the new data model are on different abstraction levels now. Some parts of the overall model are elaborated in detail (i.e. parts where business area analyses have been conducted), whereas other parts are still on a level of coarse description. There is no way of distinguishing detailed information from high-level one.

Components of the data model defined in ISP and BAA can be processed directly in the design stage where the relational model is generated from the conceptual ER model. However, the relational model is just a so-called "first-cut" model. Since ADW's derivation rules are rather stereotype, the first-cut model needs thorough revision. Generating the relational model from the ER model could be improved significantly if up-to-date research results were implemented (cf. Markowitz and Shoshani, 1992; McCormack et al., 1993; Batini et al., 1992). For example, if information about functional dependencies were attached to the ER model during analysis and used by the translator, the resulting relational model would be more adequate than the uncouth first-cut model.

When changes of the ER model are made, a new relational model (at least a partial model) has to be generated. Before that, however, all elements of the relational model affected by the modification must be deleted manually – an error-prone and time-consuming task. ADW version 2.7 has been improved in this respect. A so-called *data catalog* was introduced as a buffer between BAA and SD stages. Among others, it provides an update option that automatically deletes old information and inserts new one. Furthermore, vertical integration on the data side has been extended into construction. Relations or relational databases can be handed over directly to the *DB2 Storage Diagrammer*. This new tool can be used for tuning the databases from the design stage.

On the *activities side*, top-down forward integration from business functions to processes and further on to elementary processes is intuitive and easy, but only up to the design stage. A minor drawback of former ADW versions was that all procedural logic finally had to be specified in elementary processes. This meant that no abstraction or refinement levels were left in the final code. (In version 2.7, logic may now also be specified for processes other than elementary ones.)

Procedural logic descriptions from the Minispec Action Diagrammer can be adopted by the Module Action Diagrammer, but they serve only as comments there. In particular, there is no support whatsoever as to translation of minispecifications into module-action code. This means that the developer has to write new procedures; in doing so, the minispecifications may be taken as a guideline, or may be ignored.

In version 2.7, some improvements regarding the activities side have been made. Module action diagrams generated from BAA processes may now contain parameter lists and calls of other module actions. Parameter lists and calls are automatically derived from data-flow diagrams and from decomposition diagrams, respectively. Minispecifications are still only comments. A new feature is, however, that comments specifying the connection between a module-action diagram and the process it was derived from are inserted automatically into the module-action header.

Whereas construction is fairly efficient regarding database definitions and user-interface components, it is extremely awkward when it comes to database accesses. This is primarily due to the fact that the tools involved were adopted from ADW's mainframe predecessor and have not been sufficiently integrated into the OS/2 environment yet. In order to generate correct code, a large number of technical details have to be specified by the developer. Since error messages are insufficient, fault diagnosis is extremely difficult. Our experience showed that it is sometimes easier to delete modules, redefine them completely, and try to generate again.

As to *reverse integration*, there is hardly any support. For example, changes of module actions during design are not available in the minispecifications of BAA. If the designer nevertheless adapts minispecification code copied from analysis, modifications will not be transferred to the Analysis Workstation.

3.4 Interpersonal Integration

ADW's central encyclopedia is central only in the sense that it is used by all workstation tools. Team work is only adequately supported if the encyclopedia is on a mainframe. In this case, each developer can download parts of the encyclopedia to his PC, process them, and upload them back to the mainframe. Appropriate locking mechanisms guarantee that other developers have read-only access to the respective parts.

If no mainframe is available, the encyclopedia runs on a PC, but it is not LAN-based. Instead, ADW allows several parallel encyclopedias to be kept and consolidated from time to time. This is a rather insufficient substitute for large projects, however. Organizational rules are necessary to coordinate decentral activities.

In one of our projects outlined in Kurbel (1994), about 30 people were involved. To cope with this situation, a master encyclopedia on one computer and nine additional working encyclopedias on other computers had to be created. They were consolidated at regular intervals. Between consolidation runs, read-only copies of the master encyclopedia and of the working encyclopedias were given to the project subteams. Consolidation runs took very long; for 9 encyclopedias, they amounted to ½ - 1 day during which encyclopedias could not be used.

Recently KnowledgeWare introduced new tools that improve interpersonal integration. The *Encyclopedia Expert* simplifies consolidation of individual PC-based encyclopedias into one common project encyclopedia. The *Workgroup Coordinator* for ADW, version 2.7, allows simultaneous access to a LAN-based encyclopedia by several users.

4. INTEGRATION ASPECTS OF IEF

IEF (Information Engineering Facility) was developed by James Martin Associates Inc. (now Texas Instruments Information Engineering Inc.). IEF runs under OS/2 PM and Windows. Target environments are IBM mainframes under MVS and PCs under OS/2 PM and Windows. This evaluation is based on IEF, version 5.1, for OS/2 PM both as development and target environment.

4.1 Tools of IEF

Information collected during the stages of IE is stored in an encyclopedia whose objects are largely equivalent to ADW objects. IEF offers features to define new objects, i.e. objects not provided by IEF, and to specify interrelations between these objects. In this way, it is possible to adjust IEF to enterprise-specific information requirements.

IEF's model of IE stages comprises ISP, BAA, business system design (BSD), technical design (TD), and construction (Texas Instruments, 1993). Martin's system design stage thus was split up into two stages. TD activities are related to a particular target environment (e.g. programming language, teleprocessing monitor, database management system, etc.).

Figure 2 shows major IEF tools for the first four stages. The tool set includes tools for defining and representing matrices *(Matrix Processor)*, ER diagrams *(Data Model Diagrammer)*, and hierarchy diagrams *(Activity Hierarchy Diagrammer* and *Organizational Hierarchy Diagrammer)*. The *Activity Dependency Diagrammer* is used to model sequences of functions and processes. Procedural logic of elementary processes and procedures is specified by means of the *Action Diagrammer*. The *Structure Chart Diagrammer* is used to process calling structures between elementary processes. The *Dialog Design Diagrammer* is applied to process calling structures between procedures. With the help of the *Data Structure Diagrammer*, the relational schema generated from the ER model can be optimized. The *Window Diagrammer* enables the developer to define a graphical user interface.

4.2 Horizontal Integration

IEF's encyclopedia is also a *central* encyclopedia with respect to the tool view, but not with regard to interpersonal cooperation. IEF's concept of horizontal integration corresponds largely to the concept outlined for ADW. Objects and information about object relationships are exchanged between tools via the encyclopedia. In addition, it is possible to link information in a way that new objects will be derived automatically. This concept is illustrated by the following example.

During analysis, procedural logic of elementary processes is specified by the developer. For each elementary process, IEF generates a template that already contains import and export views, the specified data accesses (create, update, etc.), and standard exception handling. Information necessary to generate action blocks is taken from the process

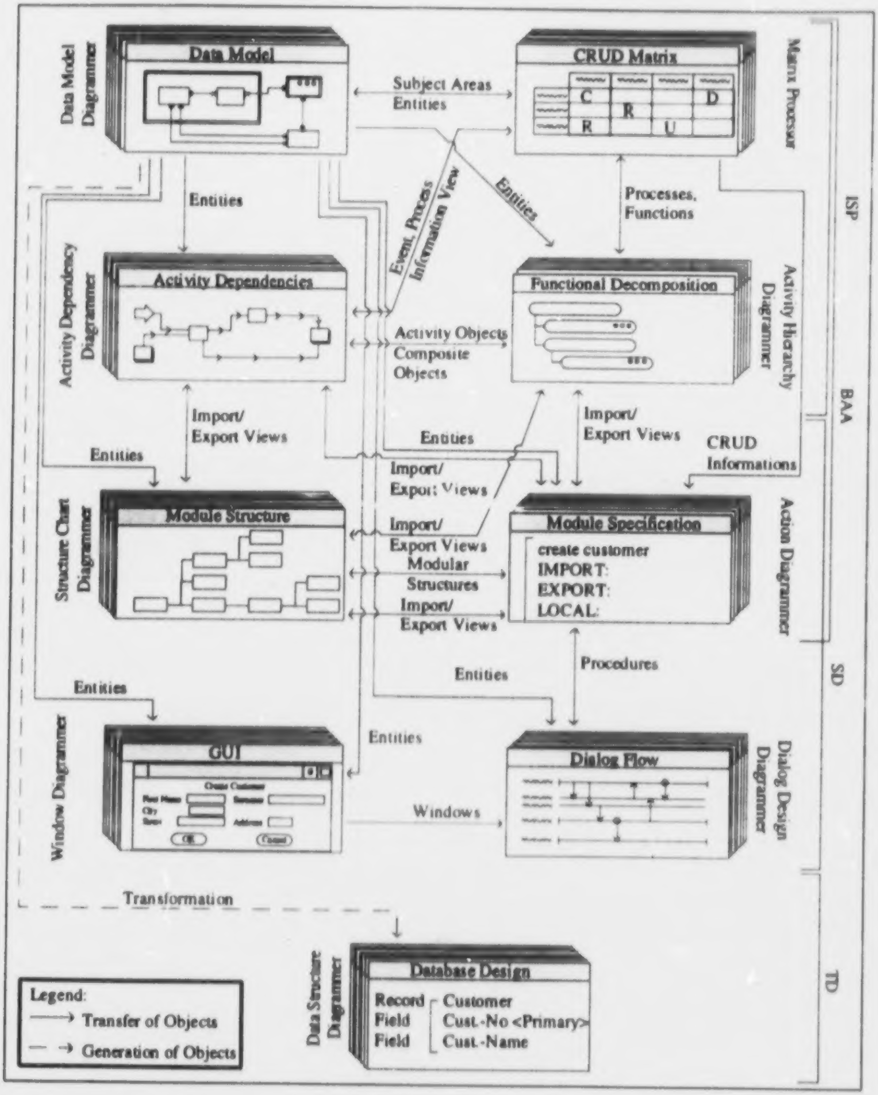

Data Model Diagrammer
Data Model
Subject Areas
Entities
CRUD Matrix
C
R
D
R
U
Matrix Processor
Entities
Event, Process Information View
Entities
Processes, Functions
ISP
Activity Dependency Diagrammer
Activity Dependencies
Activity Objects
Composite Objects
Functional Decomposition
Activity Hierarchy Diagrammer
BAA
Import/ Export Views
Entities
Entities
Import/ Export Views
CRUD Informations
Import/ Export Views
Structure Chart Diagrammer
Module Structure
Import/ Export Views
Modular Structures
Import/ Export Views
Module Specification
create customer
IMPORT:
EXPORT:
LOCAL:
Action Diagrammer
Procedures
Entities
SD
Entities
Entities
Window Diagrammer
GUI
Windows
Dialog Flow
Dialog Design Diagrammer
Transformation
Data Structure Diagrammer
Database Design
Record Customer
Field Cust.-No <Primary>
Field Cust.-Name
TD
Legend:
Transfer of Objects
Generation of Objects

Figure 2. Horizontal and vertical integration in IEF

vs. entity-type matrix or from statements of *expected effects* given by the developer. Expected effects describe how entity types of the data model are accessed (read, write, etc.). Action-block templates are then filled by the developer with the help of a specific editor that checks input for syntactical correctness.

4.3 Vertical Integration

In the planning stage, developer build a high-level overview data model of the enterprise. Components of this model are subject areas and associations between those areas. Subject areas summarize closely related entity and relationship types. In the analysis stage, the data model is refined by more detailed subject areas and by entity types. Relationships can be specified between entity types, but not between subject areas. However, associations between subject areas are automatically derived from relationships between entities that belong to different subject areas.

The high-level data model is available for later tasks. In contrast to ADW, it is not destroyed by BAA activities, but clustering of entities and relationships is not supported, either. Therefore it is not possible to represent relationships between abstract and more detailed entities.

From the BAA data model, IEF automatically derives a relational database schema. This schema may then be revised in the TD stage. Two advantages have to be emphasized: First, IEF prevents any modification of the database schema that would be inconsistent with the ER model. For example, a data type assigned to a table column cannot be modified directly. It has to be changed in the ER diagram first, before it can be adapted in the database schema (by new generation). Second, data accesses defined in action blocks refer to the ER model and not to the database schema. This means that there is no need for developers of action blocks to use identifiers from the database schema they are not familiar with (because the schema was generated!).

On the *activities side*, information processing is coherent, too. During ISP, a Hierarchy Decomposition Diagram for business functions and a Functional Dependency Diagram are developed. The respective information is specified in detail during BAA. The Hierarchy Decomposition Diagram is refined down to the level of elementary processes. Dependencies between processes are modeled by extending the Functional Dependency Diagram. Diagrams of different abstraction levels, from ISP and BAA, remain separate, as on the data side. Thus the ISP model may be reused for other BAA projects.

Procedural logic of elementary processes is described by means of the Action Diagrammer during BAA. In the SD stage, IEF automatically assigns a procedure to control user interaction with the program to each window or screen. To fill such a procedure, the developer specifies which action blocks have to be executed when certain user inputs occur. For example, he specifies that the elementary process "issue and print invoice" will be executed if the user enters the command "issue invoice". A *procedure action block* containing the calls of the corresponding action blocks is generated automatically from this information. Additional procedural logic may be filled in by the developer.

IEF's concept of vertical integration on the activities side has two major advantages: First, descriptions of elementary processes from BAA can be used for SD. (In contrast, ADW requires entirely new editing of module actions in SD). Second, modifications of action blocks are automatically available for tools of later stages.

It should be emphasized, however, that IEF's concepts for the activities side do not support top-down development well. On the contrary, procedural logic of elementary processes has to be described on a formal language level in BAA already. In addition, procedure action blocks usually are on a higher abstraction level than action blocks. Since they are specified *after* elementary processes (action blocks) have been developed, IEF rather supports a bottom-up approach. Furthermore, elementary processes and procedures may be difficult to understand because abstract descriptions of procedural logic do not exist.

Before code is generated in the construction stage, IEF automatically checks consistency of the models developed in the previous stages. Source code is only generated if no errors are detected. In contrast to ADW, the generated source code is free of syntax errors. A debugger on the *design level* helps to search for semantic errors. If such a debugger is not available (as in ADW), developers have to examine error messages referring to the generated source code (which they are not familiar with).

IEF does not support *reverse integration*. Some of the problems, however, are totally avoided because of rigorous restrictions. First, modifying objects created in one stage later on is either prohibited or only permitted if it cannot cause inconsistency. One example was mentioned before: The database schema generated from the BAA data model can only be modified in TD if changes do not affect consistency with the ER model. Second, in many cases there are just no means to describe the same things on different abstraction levels. Therefore there is no need to update models of earlier stages. On the activities side, for example, procedural logic of elementary processes has to be specified completely on the language level of action blocks. More abstract descriptions do not exist. Further refinement of action blocks by SD tools is not possible. Action blocks may only be called from procedures developed in SD.

4.4 Interpersonal Integration

IEF's support for interpersonal integration has three aspects. First, the encyclopedia can be stored on a mainframe. Using it is then similar to ADW. Second, the encyclopedia of version 5.1 underlying this paper does not support team work at all. Third, a client-server encyclopedia supporting LAN-based team work has recently become available, but we have not evaluated this version yet.

5 SUMMARY

In this paper, integration-related aspects of two IE-based I-CASE were elaborated as to how well they support horizontal, vertical, and interpersonal integration.

From our experience with ADW, horizontal integration is excellent, but vertical integration has quite some flaws. With regard to data modelling, integration is quite satisfactory across ISP and BAA stages, but integration between BAA and SD needs to be improved considerably. On the other hand, the activities side suffers severely from lack of integrative features. Interpersonal integration was also weak up to now and has only recently been improved. The old concept of consolidation across encyclopedias was insufficient for efficient support of project work. Our experience with consolidation in a large project was presented in Kurbel (1994). The Workgroup Coordinator promises to remedy this drawback.

IEF tools, compared to ADW, appear to be better integrated. Whereas horizontal integration is about the same, vertical integration has significant advantages over ADW's concepts. Not only the tools for data modelling are integrated across stages, but the tools creating successive representations of activities (processes, procedures, programs) are integrated, too. Program-construction and design tools are integrated better than in ADW. This is particularly useful when it comes to debugging. In addition, IEF prevents some problems from missing reverse integration although the way it is done is rather restrictive.

References

Batini, C., Ceri, S. and Navathe B., (1992), "Conceptual database design: an Entity-Relationship Approach", Benjamin/Cummings, Redwood City, CA.

Bucken, M., (1992), "AD/Cycle Changes Continue", *Software Magazine*, Vol. 12, No. 14, pp. 26-30.

Dineur, A., (1990), "ATMOSPHERE: CASE for System Engineering Support", *in*: "Fourth International Workshop on Computer-Aided Software Engineering", R. J. Norman and R. Van Ghent, eds., IEEE Computer Society Press, Los Alamitos et al.

Jaeschke, P., Oberweis, A. and Stucky, W., (1994), "Extending ER Model Clustering by Relationship Clustering" (to appear).

KnowledgeWare Inc., (1994), "Application Development Workbench - Overview", 0194-PU-0665, KnowledgeWare Inc., Atlanta, GA.

Kramer, J., (1991), "CASE Support for the Software Process: A Research Viewpoint", *in*: "Proceedings of the 3rd European Software Engineering Conference", Lecture Notes in Computer Science 550, A. Van Lamsweerde et al., eds., Springer Verlag, Berlin et al.

Kurbel, K., (1994), "From Analysis to Code Generation: Experiences from an Information Engineering Project Using I-CASE Technology", *in*: "Advanced Information Systems Engineering", 6th International Conference, CAiSE '94, Proceedings, Lecture Notes in Computer Science 811, G. Wijers et al., eds., Springer-Verlag, Berlin et al.

Garbajosa, J. et al., (1990), "Position Paper: Implementing Cooperation and Coordination in Software Engineering Environments", *in*: "Fourth International Workshop on Computer-Aided Software Engineering", R. J. Norman and R. Van Ghent eds., IEEE Computer Society Press, Los Alamitos et al.

Markowitz, V. M. and Shoshani, A., (1992), "Representing Extended Entity-Relationship Structures in Relational Databases: A Modular Approach", *ACM Transactions on Database Systems*, Vol. 17, No 3, pp. 423-464.

Martin, J, (1989), "Information Engineering", Book I Introduction, Prentice-Hall, Englewood Cliffs.

McCormack, J. I., Halpin, T. A., Ritson, P. R., (1993), "Automated mapping of conceptual schemas to relational schemas", *in*: "Advanced Information Systems Engineering", 5th International Conference, CAiSE '93, Rolland, R. et al., eds., Lecture Notes in Computer Science 685, Springer-Verlag, Berlin et al.

Nomina Gesellschaft für Wirtschafts- und Verwaltungsregister, (1993), "ISIS PC Report, Software für Personal Computer & PC-Netzwerke 2", Vol. 2, Nomina Gesellschaft für Wirtschafts- und Verwaltungsregister mbH, München.

Polilli, S., (1992), "IBM Redirects AD/Cycle", *Software Magazine*, Vol. 12, No. 12, pp. 31-34.

Rauh, O. and Stickel, E., (1992), "Entity tree clustering - A method for simplifying ER design, *in*: "Proceedings of the 11th International Conference on Entity-Relationship Approach", G. Pernul, A.M. Tjoa, eds., Springer-Verlag, Berlin et al.

Robinson, D., (1991), "CASE Support For Large Systems", *in*: "Proceedings of the 3rd European Software Engineering Conference", Lecture Notes in Computer Science 550, A. Van Lamsweerde et al., eds., Springer Verlag, Berlin et al.

Stobart, S. C. et al., (1993), "An Empirical Evaluation of the Use of CASE Tools", *in*: "Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering", H.-Y. Lee et al., eds., IEEE Computer Society Press, Los Alamitos et al.

Teorey, T. J. et al., (1989), "ER model clustering as an aid for user communication and documentation in database design", *Communications of the ACM*, Vol. 32, No. 8, pp. 975-987.

Texas Instruments Inc., (1993), "Developer's Reference", 2632047-7301, Texas Instruments Inc., Plano, Texas.

Venable, J. R., (1990), "Achieving Effective Tool Integration in a CASE Environment", *in*: "Fourth International Workshop on Computer-Aided Software Engineering", R. J. Norman and R. Van Ghent, eds., IEEE Computer Society Press, Los Alamitos et al.

Wybolt, N., (1991), Perspectives on CASE Tool Integration, *Software Engineering Notes*, Vol. 16, No. 3, pp. 56-60.

Zarrella, P. F., (1990), "CASE Tool Integration and Standardization", *in*: "Fourth International Workshop on Computer-Aided Software Engineering", R. J. Norman and R. Van Ghent, eds., IEEE Computer Society Press, Los Alamitos et al.

443

AN EMPIRICAL STUDY OF KEY FACTORS IN CASE IMPLEMENTATION

Erna Rupnik-Miklič[1], Jože Zupančič[2]

[1]Statistical Office of Republic Slovenia, SI-61000 Ljubljana, Vožarski pot 12, Slovenia
[2]University of Maribor, School for Organizational Sciences, SI-64000 Kranj, Prešernova 11, Slovenia

Abstract

Results of an empirical study of the key success factors for implementation of CASE technology in Slovenia are presented. Adequate education and knowledge of the staff was identified as the most important among the 18 listed factors. With exception of the selection of the appropriate CASE tool, people and management related issues (such as use of development methodology, existence of IS development standards, project management and management of expectation) were ranked the highest. Results of this study also demonstrate that preparation for implementation of CASE, such as training of IS professionals in methodology and in the usage of the tool, an explicitly defined plan for implementation and having a system development methodology in place prior to the implementation of CASE positively influence the expectation fulfilment and therefore the satisfaction of CASE users.

1. INTRODUCTION

While interest and investment in Computer-Aided Software Engineering (CASE) tools have been rising steadily, actual experiences with tools have exhibited more ambiguity. Some studies report improvements in productivity from the use of CASE, others find that expected productivity gains are elusive or eclipsed by lack of adequate training and experience, developer resistance, and increased design and testing time (Orlikowski, 1993). Yet, users have been slow to adopt the tools and benefits have been equally slow to appear. A few organisations have achieved benefits beyond expectations, others are not sure about the true costs and returns on CASE tools investments or are even dissatisfied with their current CASE implementation. And yet other organisations are evaluating or considering the acquisition of CASE technology, many of them intending to implement CASE if and when they find an appropriate tool.

A number of studies dealing with implementation of CASE have been published in recent years. Parkinson (1990) presented an overview of critical success factors and discussed a number of common experiences that point at reasons why CASE has been slow to be adopted: confusion about what a CASE tool is, over-sold capabilities, changes in working methods, too little initial vendor support, underestimation of learning curve effects, poor match of the selected tool to the audience and the critical mass effect. Due to learning curve effects, productivity usually declines below established rates for some time. There is yet no standard way to determine the length of that learning period. Estimates range from six months to two years (I/S Analyzer, 1993). Wynekoop and Senn (1992) regard, the implementation of CASE technology in an organization as a change affecting the organisation as well as individuals, and which must, therefore, be properly planned and managed. Alavi (1993) recommends the use of team oriented tools and alignment of CASE methodology and software development work patterns as well as adequate training and management support. Hruschka (1991) discussed the most frequently stated reasons for not implementing CASE technology: inability to measure or confirm the benefits of CASE, high cost, lack of development standards, limitations of developer creativity, and unsuccessful attempts at implementation. He emphasized that successful adoption of CASE demands changes in problem solving methods, which are much more difficult and take much more time and effort than changes in programming habits, documentation, hardware, programming language or organisation. McChesney and Glass (1993) suggest that the post-implementation stage is the most critical for ensuring successful use and incorporation of CASE methodology within an organisation and propose a framework for its post implementation monitoring and management as a socio-technical innovation.

Virtually all authors emphasise the importance of adequately educating and training developers in both tools and underlying methods, the existence of methodological development standards, and management commitment to, and support of, CASE adoption. In spite of the relatively slow implementation of, and reported resistance to CASE infusion in many companies, most authors however expect the use of CASE tools to lead to increased staff productivity and to the generation of quality software.

2. FRAMEWORK AND GOALS

Recent developments in Central and Eastern European countries seem to increasingly interest the information technology (IT) community. Kempfer (1993) expects the value-added services in Eastern Europe to expand at an annual growth of approximately 20% during the next five years. Dyson (1993) identified two groups of Central and Eastern European countries; she grouped Slovenia with Hungary, the Czech Republic, and Poland as the four countries leading in the level of IS activities and industry.

In Slovenia, computing and the application of IT have developed at a considerable rate, as organisations of many kinds and sizes have adopted or are in the process of adopting computerised IS. A comparison of major issues in IS management in Slovenia and in the USA (Dekleva and Zupančič, 1993) reveals a high presence of management-related issues, particularly issues internal to IS organisation, in comparison to USA. The study also indicates that IS-related services, such as educational, legal and professional services, are still scarce or inefficient in Slovenia, yet they do seem to be improving. The same can be said for the national information infrastructure, regarding for example communication facilities, product supply, and external databases. Among the system development-related issues, the use of integrated IS development methodology and the application of development tools, such as CASE tools and 4GLs, were rated as the most important.

This paper presents the results of a survey carried out in Slovenia in January 1993. Its aims were the following:

1. To identify and evaluate key factors that impact CASE implementation in organisations.

2. To determine how to what extent the following factors are correlated to the expectation fulfilment of CASE users:
- the amount of CASE-related education and training prior to the implementation of the tool
- the existence of a plan for CASE implementation
- the use of system development methodologies prior to CASE implementation, and
- the duration of CASE use

The findings reported in this article are also believed to be indicative for other new Central and East European democracies -- particularly Hungary, the Czech Republic and Poland, which are at a comparable stage to Slovenia in adopting IT and developing their IS technology markets.

3. SURVEY DESIGN AND RESPONDENT PROFILE

The Governmental Office for Statistics provided a list of 300 industrial, commerce and service organisations with more than 300 employees. Because major political and economic changes initiated recently have caused dynamic restructuring of businesses, many organisations were forced to disintegrate, downsize, or restructure to compete in new markets. Although outdated, the list used for this study was the best available at the time.

In designing the survey questionnaire, a pilot survey among a limited number of respondents was undertaken that used a draft questionnaire to be completed and evaluated. Where possible the questionnaire included multiple choice questions with space to collect additional information if and when it was needed. In other questions respondents rated the importance (or frequency, proportion, etc.) of the listed items. The questionnaires was sent to managers of IS Departments. Usable responses were obtained from 102 firms, yielding a response rate of 34%. Among them, 22 (21.6%) actually used CASE. In order to orient the respondents properly, the following definition of a "user" was included in the questionnaire:

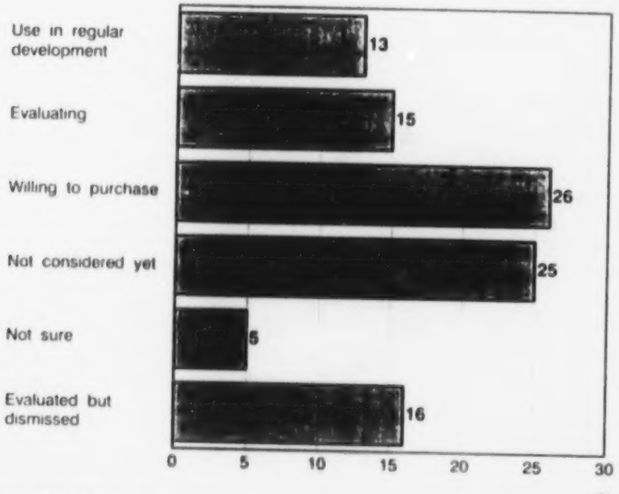


Figure 1 Use of CASE for 102 respondents

CASE users (CASE respondents) are organisations where at least one real-world development project with a minimum size of 6 man-months had been completed using a CASE tool. Figure 1 shows the use of CASE by the respondent organisations.

The majority of CASE respondents (13 out of 22) had completed with aid of CASE at least one project of a minimum size of 6 man-months and had regularly used the tool in their development work. However, 6 out of 15 organisations who responded that they were evaluating CASE had purchased a CASE tool and had already completed at least one such project, but were not yet using CASE regularly. A considerable number of non-CASE respondents (27) were willing to buy CASE if and when they found an appropriate tool, 26 had not yet considered or evaluated CASE, and 5 were not sure whether CASE would be useful for their organisation. Among 16 organisations who had evaluated but not adopted CASE, 3 organisations had completed at least one 6 man-month project using CASE and temporarily abandoned CASE; they were also considered CASE respondents.

The portion of CASE respondents cannot be considered representative for the whole population as reasons for non-reply were not thoroughly investigated. A recent investigation, though not focused on CASE (Verber, 1993), found that of 120 organisations with a total number of employees more than 300, 12% have purchased a CASE tool.

Respondents who supplied data came from a wide variety of business backgrounds and from organisations of all sizes. The average number of employees was 1080 although only 30 companies employed more than 1000 people, with the largest organisation having 6200 employees. No major differences in the total number of employees between CASE and non-CASE respondents were found, but IS Departments of CASE respondents were generally larger than those of non-CASE respondents. Most of the CASE respondents (71%) used project team organisation and employed it in all large development projects, compared to less than half (43%) of non-CASE respondents.

4. EXPERIENCES AND EXPECTATIONS WITH CASE TECHNOLOGY

Organisations using CASE are characterised by, relative to their peer organisations, a higher level IS maturity, where the diffusion of the use of system development methodologies, structured techniques, strategic planning and project management are significant, and where some corporate development standards are already in place. None of the sampled organisations adopted CASE as the standard way to develop and maintain software. Although in Slovenia, as in other Central and East European countries, available funds for investments are limited and hinder wider implementation of advanced IT, lack of related methodological and management knowledge and skills was indicated by the respondents as the major reason for non- or slow adoption, and then limited use, of CASE by system developers. In Slovenia, 42% of respondents of 80 non-CASE respondents gave lack of knowledge and skills as the major reason, and 23% the high cost of the tool. An investigation in UK showed, that high cost was the major reason for non-adoption of the tool (31% of respondents), while lack of knowledge and skills was not considered as the most important reason by any of the 118 UK respondents (Stobart et al. 1991).

The extent to which expectations of CASE users regarding the benefits of CASE were met varies widely. Ratings were the highest for a disciplined and methodological approach and for orderly and transparent system development procedures. Realisation of expectations that many CASE vendors emphasise the most, such as higher productivity, shorter system development time and automation of work, ranked the lowest. Major deficiencies identified by CASE users were poor or missing code generators and poor integration with other tools. Regarding the features and facilities that should be provided by future tools, IS Department managers in Slovenia clearly favoured code generators and reverse engineering facilities -- facilities that they have had the least opportunity to work with, as most of the tools available had no efficient code generators and no embedded re-engineering components.

Adoption and current use of CASE technology, perceived benefits, the expectations and major differences in the approach to IS development between organisations that currently use or do not use CASE are discussed in (Rupnik, 1993) in some detail.

5. KEY FACTORS AND PREREQUISITES FOR CASE IMPLEMENTATION

To determine the key factors that impact successful implementation of CASE we prepared a list of 18 factors identified by several previous studies (e.g., Parkinson 1990, Hayley and Lyman, 1990, Wynekoop and Senn 1992) and by a preliminary study. We asked the respondents to rate the significance of 5 to 8 factors by assigning weighted values to selected items so that the total number of points totalled 100. They were asked also to indicate factors that they considered necessary prerequisites for successful implementation of CASE in the system development process. Table 1 presents the summary of their ratings, the column "Prerequisites" shows the percent of respondents who considered the listed factor as a prerequisite for successful CASE introduction.

Table 1: Key success factors for CASE implementation

Factor	*Average weight*	*Prerequisite*
1. Adequately educated IS personnel	15.7	86%
2. Selecting appropriate CASE tools	10.0	59%
3. Structured development methodology in place	8.4	82%
4. IS development standards in place	8.4	64%
5. To know exactly what you expect from CASE	8.4	55%
6. Training in CASE tool usage	7.7	73%
7. Awareness of the need for a systematic development approach	6.2	50%
8. Decision of the management to adopt CASE	6.2	23%
9. Project team work and good project management	4.8	68%
10. Involvement and motivation of end users	4.4	18%
11. Senior management commitment	3.6	55%
12. Strategic IS development plan in place	3.4	23%
13. Appropriate hardware	3.4	32%
14. Explicitly defined strategy (plan) for CASE implementation	2.7	41%
15. Integration of the tool into the existing development environment	2.5	9%
16. Consulting support	1.6	5%
17. Information literacy of end users	1.6	5%
18. Understanding of the impact of CASE technology	0.9	14%

CASE respondents in our study considered adequate education of the staff as the most important success factor, which is compatible with the opinions of non CASE respondents to whom lack of knowledge and skills was the most important reason for non-implementation of CASE. The importance of knowledge in skills, specifically knowledge of the underlying methodology has been emphasised also in other studies (Hayley and Lyman 1990, Parkinson 1990, Wynekoop and Senn, 1992). Except the selection of the appropriate CASE tool, people and management related issues, such as use of development methodology, existence of IS development standards, project management and management of expectation, were ranked the highest. Some factors, such as explicitly defined strategy for CASE implementation and good management of the implementation, integration of the tool into the existing development environment (rather than radically changing it), external consulting support, and understanding the impact of CASE technology in order to prevent unrealistic expectations, are ranked relatively low, although many authors (e.g., Parkinson 1990, Alavi 1993) consider them among the most important ones. This may be explained by relatively small size of companies that participated in our study and a small number of developers who use CASE (maximum 6 for 2 companies, while only one developer used CASE in 6 out of 22 CASE respondent organisations), and by the relatively short time of using the tool.

When respondents were asked to specify major barriers for implementation of CASE again most of them suggested insufficient knowledge and skills, in particular knowledge of development methodologies. Other stated factors such as missing standardisation of the development process and non-adhering to standards, poorly defined business processes in the firm, and insufficient motivation.

In the USA 64% of companies used consulting support in 1992 in the following IS-related areas of expertise: re-engineering, downsizing, outsourcing open systems and system development, where the largest number of days of consulting was in system development and CASE tools (ComputerWorld, 1993). Our study showed relatively low level of satisfaction and limited use of consulting support for CASE implementation: Among companies surveyed 7 (32%) used an average of 5.4 days of external; three of them found it unsatisfactory. Most of the respondents who did not use external consulting (41%) felt a need for it or believe that there is no adequate consulting support available (18%); only two organisations (9%) felt no need for consulting.

6. IMPACT OF PREPARATION FOR CASE IMPLEMENTATION ON EXPECTATION FULFILMENT

6.1 Education and training

Most of the CASE respondents in our study (64%) provided internal or external training to developers who were responsible for implementation and use of CASE both in the use of tool and underlying methodology, while 4 (18%) provided training only in the use of the selected tool, 2 (9%) only in the methodology and 2 (9%) organisations provided no preparation (Figure 2). The average duration of training was 4.4 days for the use of the tool and 6.9 days for methodology per participant.

We tested the hypothesis that duration of training in CASE tool usage and development methodology increases expectation fulfilment of their users. The χ^2 test showed a statistically significant correlation between the duration of training per developer and the estimated level of expectation fulfilment ($p=0.025$).

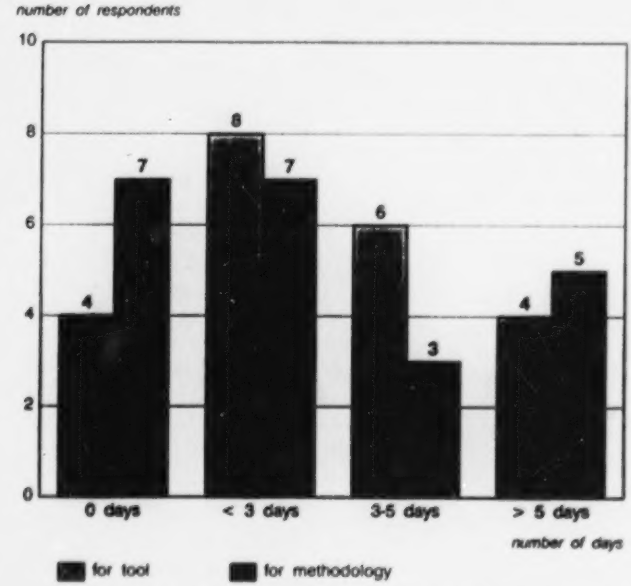


Figure 2: Training for the use of the tool and in methodology

6.2 Strategic plan for CASE implementation

While 41% of respondents considered an explicitly defined plan as a prequisite for successful CASE implementation, they indicated the following levels of planning for CASE implementation in their organisations:

2 (9%) - a precise written plan for CASE implementation was elaborated
8 (36%) - CASE implementation was included in the plan, but was not precisely defined
5 (23%) - only some written guidelines for CASE implementation were prepared
7 (32%) - no written plan existed, CASE implementation was only discussed

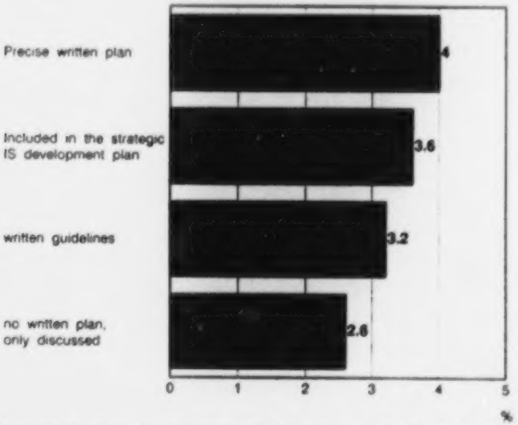


Figure 3: Expectation fulfilment depending on the level of planning for CASE implementation

Figure 3 presents the rated level of expectation fulfilment depending on the level of planning. The hypothesis that expectation fulfilment in organizations that had a detailed plan for introduction of CASE was higher than in others was tested. The χ^2 test showed a statistically significant correlation between the level of planning for CASE implementation and the expectation fulfilment of the respondents ($p=0.05$).

6.3 Use of system development methodologies

Respondents indicated the following level of the use of system development methodologies prior to the implementation of CASE:

3 (14%) - for full system development life cycle (SDLC) for most applications
10 (45%) - for some phases in the SDLC for the majority of applications
1 (5%) - for the whole SDLC in some applications
6 (27%) - for some phases in the SDLC for some applications
2 (9%) - no methodology was used

Figure 4 gives the rated level of expectation fulfilment depending on the level of methodology use. We tested the hypothesis that the expectation fulfilment of users who use a development methodology prior to implementation of CASE is higher than in others. The χ^2 test showed a statistically significant correlation between the level of methodology use and expectation fulfilment of the respondents ($p=0.01$).

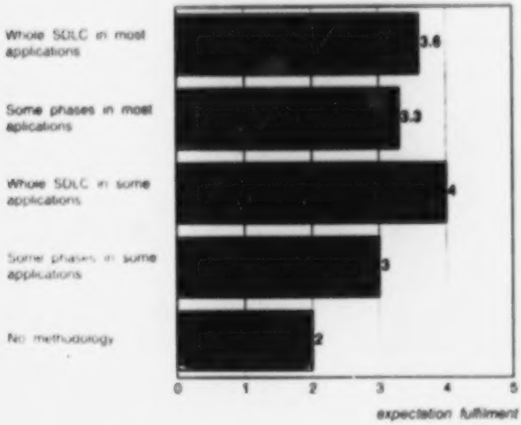


Figure 4: Expectation fulfilment and use of a development methodology prior to CASE implementation.

6.4 The duration of CASE use

Results of empirical investigations (e.g., Aaen et al., 1992) indicate that with prolonged use of CASE technology in regular development work the satisfaction of its users increased. Figure 5 shows the duration of use of CASE tool in the surveyed organisations, and Figure 6 the rated expectation fulfilment. Due to the small sample size a statistically significant correlation between the length of use and fulfilment of expectations could not be confirmed.

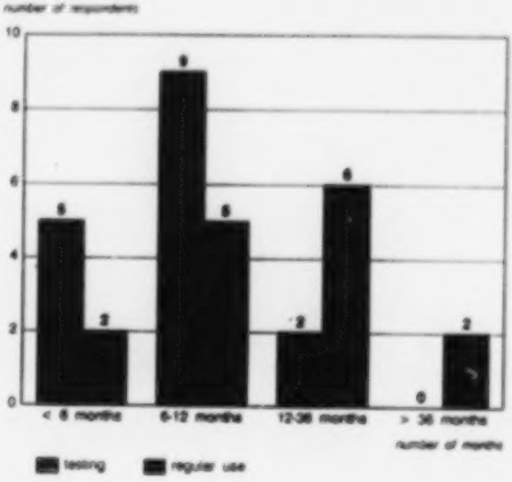


Figure 5: Duration of test regular use of CASE

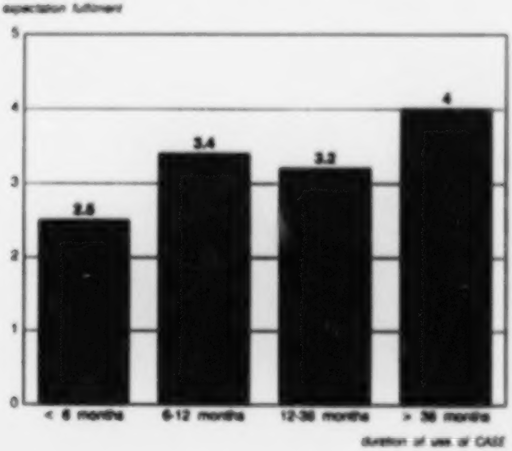


Figure 6: Expectation fulfilment depending on the duration of CASE use

7. CONCLUSION

Although a good and appropriate CASE tool was found important for successful CASE implementation, adequate education of the staff was identified by CASE respondents in our study as the primary success factor. On the other hand, non-CASE respondents most frequently stated insufficient knowledge as the major reason for not adopting CASE. This matches with the findings of a recent comprehensive study of key issues in IS management where the education of IS professional was identified as the second most important factor, preceded only by inadequate appreciation of IS by executives and other users (Dekleva and

Zupančič, 1993). Adequate education of the staff can be related to knowledge and use of structured methodology, which was rated as the third most important factor, while training in using the selected tool was considered relatively less important and ranked in sixth place. Results of our study also demonstrated that preparations for the implementation of CASE, such as training of IS professionals in methodology and in the usage of the tool, an explicitly defined plan for implementation and a system development methodologies in place positively influence the expectation fulfilment and therefore the satisfaction of CASE users. Although the importance of external consulting support was rated very low, only a few respondents felt no need for it, about one third used it, and about a half of these who used consulting found it unsatisfactory..

Understanding the key success factors for CASE implementation should help IS executives and managers to focus and direct their attention. Therefore, results of this study may be relevant for organisations considering the implementation of CASE technology. They may in particular help to understand that the implementation of CASE represents a major organisational change which must be properly planned and managed to avoid disappointment, ineffective investments and failures when introducing this new technology.

REFERENCES:

Aaen, I., Siltanen, A., Sorensen, C. and Tahvanainen, V.-P., A tale of two countries: CASE expectations and experience, *The impact of Computer Supported Technologies on Information System Development*, Editor K.E. Kendall, Elsevier Science Publishers B.V. 1992, pp. 61-93.

Alavi, M., Making CASE on organisational reality -- strategies and new capabilities needed, *Journal of Information Systems Management*, Spring 1993, pp. 15-20

ComputerWorld, January 11, 1993.

Dekleva S., Zupančič J., Key issues in information systems management: a Delphi study in Slovenia," *Proceedings of the Fourteenth International Conference on Information Systems*, Orlando (Florida), December 5-8, 1993, Eds. J.I. DeGross, R.P. Bostrom, D. Robey), pp. 301-313.

Dyson, E., How Eastern Europe is starting over, *Datamation*, Vol. 39, No. 5, March 1, 1993, pp. 67-70.

Hayley K.J., Lyman. H.T., The realities of CASE, *Journal of Information Systems Management*, Vol. 7, No 3, Summer 1990, pp. 18-23.

Hruschka, P., CASE einfueren ist mehr als ein Werkzeug kaufen, in: *CASE in der Anwendung*, (Ed. P. Hruschka), Hanser Verlag, Muenchen-Wien 1991.

I/S Analyzer, Vol. 31, No. 6, June 1993.

Kempfer, L., Winds of change, *CAE*, Vol. 12, No. 2, Feb. 1993, p. 82.

McChesney, I.R. and Glass, D., Post-implementation management of CASE methodology, *European Journal of Information Systems*, Vol. 2, no. 3, 1993, pp. 201-209, May/June 1991, pp. 30-31.

Orlikowski W.J., CASE tools as organisational change: investigating incremental and radical changes in system development, *MIS Quarterly*, Vol. 17, 1993, pp. 309-340.

Ott, H.-J., Wirtschaftlichkeitanalyse von EDV-Investitionen mit dem WARS-Modell am Beispiel der Einführung von CASE, *Wirtschaftsinformatik*, Vol. 35, No. 6, 1993, pp. 522-531.

Parkinson, J., Making CASE work, in: Spurr K., Layzell P., *CASE on Trial*, John Wiley 1990, pp. 213-241.

Rupnik, E., Prerequisites and critical success factors in CASE implementation, University of Maribor, Master Thesis, 1993 (in Slovene).

Stobart, S.C., Thompson, J.B. and Smith, P, Use, problems and future direction of computer aided software engineering in UK, *Information and Software Technology*, Vol.33, No. 9, 1991, pp 629-636.

Stobart, S.C., van Reeken, A.J., Low. G.C., Trienekens, J.J.M., Jenkins, J.O., Thompson, J.B. and Jefrey, D.R., An empirical evaluation of the use of CASE tools, *Proceedings of the Sixth International Workshop on Computer-Aided Software Engineering*, Singapore, 19-23 July 1993, pp. 81-87.

Verber, B., An analysis of needed knowledge and skill for IS professionals in Slovenia, University of Maribor, Master Thesis, 1994.

Wynekoop J.L. and Senn J.A., CASE implementation: The importance of multiple perspectives, *Proceedings of the 1992 ACM SIGPR Conference*, (Ed. A.L. Lederer), pp. 63-73, ACM Press 1992.

A CASE TOOL FOR RE-ENGINEERING CONDITIONAL LOGIC

B Cowan

Napier University, 219 Colinton Road, Edinburgh EH14 1DJ, UK

Abstract

A CASE tool is described which simplifies, optimises and facilitates maintenance on complex nested IF logic statements. The tool uses reverse engineering to create a decision table from the code, automatically expand, provide editing facilities, then automatically simplify this table, and convert to best code.

1. INTRODUCTION

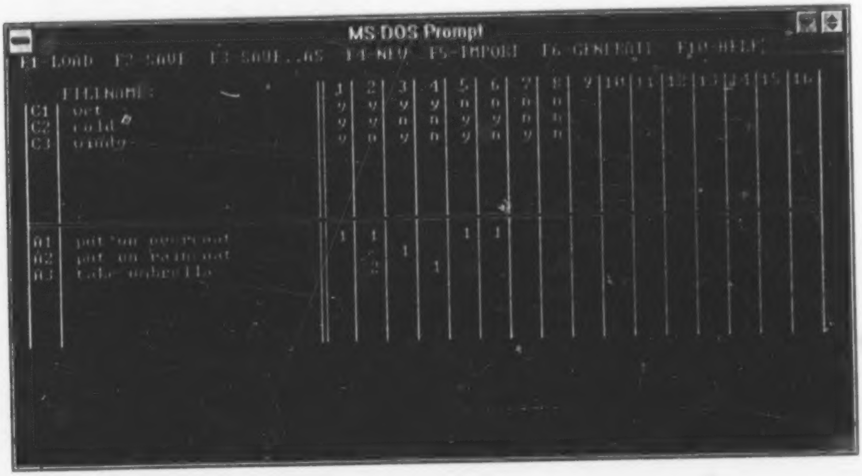
A common problem in computing is that of dealing with complex nested logic involving If .... Then .... Else(or Otherwise) .... Endif statements. It can be difficult to demonstrate that such code is correct in covering all combinations of conditions, does not have redundant statements, and does not have conflicting statements. Because there can be many ways of stating the same logical condition tests and actions, finding the optimum, simplest, clearest way may be by trial and error. Whenever an amendment is required in such logic, say to add a new condition, remove a now redundant condition, or change the actions on combinations of conditions, the logic changes and testing can be time consuming and error prone.

The computer aided tool described here simplifies the tasks of maintaining and optimising logic. As presented here it is not language specific, with the examples being in Structured English. A prototype of the tool has been constructed.

The tool operates by converting a section of logic to a decision table, allowing editing of the decision table, automatically simplifying the decision table, then regenerating logic statements from the decision table.

2. DECISION TABLES

Decision Tables are described in many books (Gane,1979; Hurley, 1983), and are outlined in this section. A decision table comprises four quadrants as illustrated by figure 1. The top left quadrant lists all the condition tests involved in the logic, and the lower left all the actions which can be taken.


MS DOS Prompt
F1-LOAD F2-SAVE F3-SAVE..AS F4-NEW F5-IMPORT F6-GENERATE F10-HELP

	FILENAME:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	wet	y	y	y	y	n	n	n	n								
C2	cold	y	y	n	n	y	y	n	n								
C3	windy	y	n	y	n	y	n	y	n								
A1	put on overcoat	1	1			1	1										
A2	put on raincoat			1													
A3	take umbrella		2		1												

Figure 1 - Example of Decision Table

The upper right quadrant contains columns showing all combinations of 'Y' or 'N' indicating whether the conditions are true or untrue. In the full decision table format there will be 2^n columns of combinations of conditions where 'n' is the number of conditions listed. The lower right has digits indicating the order of actions appropriate to the combination of conditions. Decision tables can be simplified according to rules such as:

1. Combine two columns which have the same actions, but only if there is one difference in 'Y' or 'N', and replace the 'Y'/'N' with '-' (ie the condition does not matter).
2. Delete any condition row if all columns contain '-'.
3. Delete any column if no actions are present.
4. If several condition columns contain the same set of actions, then they can be combined into an 'ELSE' column. (This can only be done once).

Simplifying the table in figure 1 according to these rules gives the table shown in figure 2. (Simplifications 2 and 3 do not apply here).

The algorithms involved are discussed by Celko (1983) and McCarthy (1986).

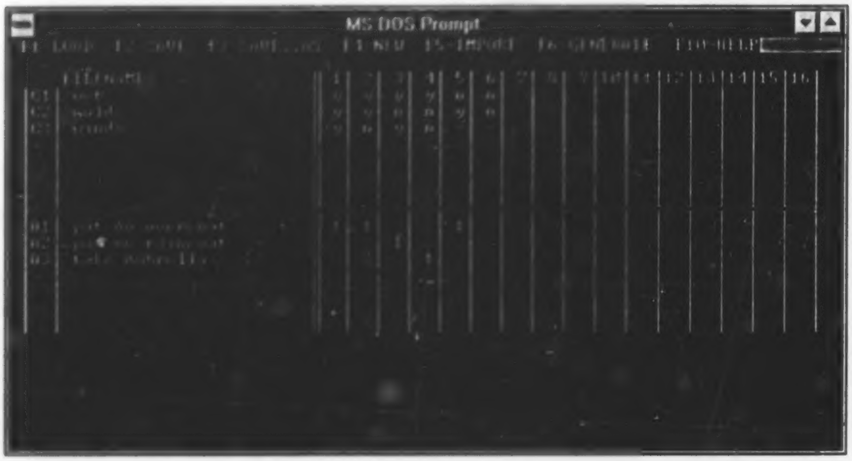


FIGURE 2 - Example Simplified

Software which converts decision tables to code has existed since the days of punched cards, but recent on-line tools have become available both within analyst/designer CASE workbenches (Computerworld, 1988) and as stand alone packages such as Logic Gem (Schulman, 1989; Coffee, 1991) and others (Boggio-Togna,1991). The novelty of the tool being described here lies in the parsing of a section of existing code to form a decision table.

Decision tables represent the logic in a non-procedural form and as such bear comparison to expert systems (Minasi, 1990). Studies by Vessey (1986) and Subramanian (1992) have shown that decision tables are more easily understood than code (or structured English), but perhaps not as clear as decision trees.

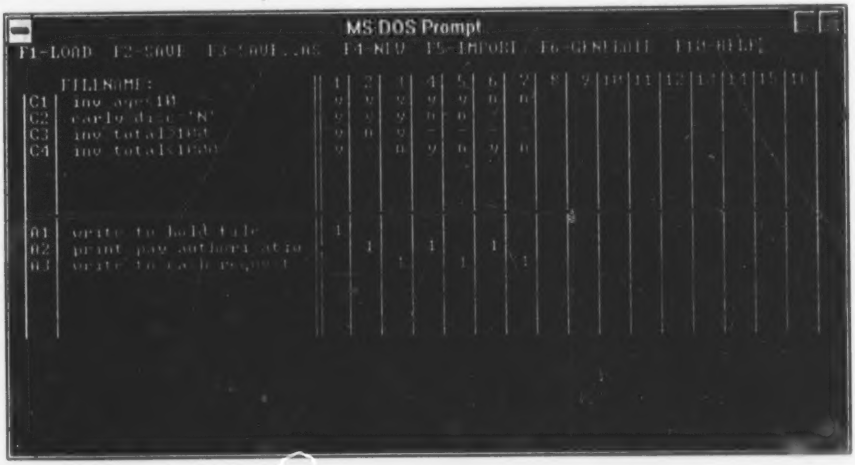
3. EXAMPLE OF TOOL OPERATION

Figure 3 gives logic relating to the payment or otherwise of invoices received by an organisation. Payment depends on the age of the invoice (inv_age), the value of the invoice (inv_total), and whether any discount is available by paying promptly (early_disc). Depending on combinations of these factors, the invoice can be paid (print pay authorisation), further authorisation sought (write to cash_request) or retained to be looked at again in another processing cycle (write to hold_file).

```
IF inv_age < 10
        THEN IF early_disc = 'N'
                        THEN IF inv_total > 100 AND < 1000
                                        THEN write to hold_file
                                ELSE  IF inv_total < 100
                                                THEN print pay_authorisation
                                        ELSE write to cash_request
                ELSE  IF inv_total < 1000
                                THEN print pay_authorisation
                        ELSE write to cash_request
ELSE  IF inv_total < 1000
                THEN print pay_authorisation
        ELSE write to cash_request
```

Figure 3 - Typical Problem Code

The tool will scan the logic to identify and list the condition tests and actions in the decision table - see figure 4. Actions under combinations of conditions are then determined by the tool and entered. Any logic conflicts or indeterminism will be detected at this stage.

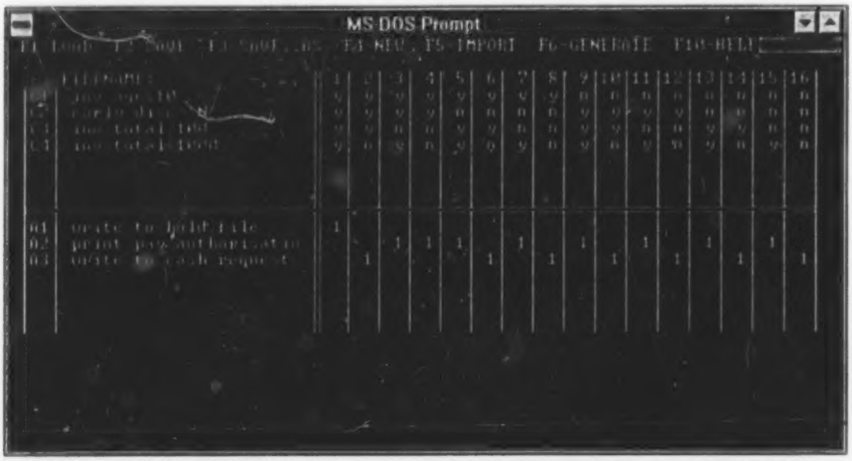

MS-DOS Prompt
F1-LOAD F2-SAVE F3-SAVE..AS F4-NEW F5-IMPORT F6-GENEDIT F10-HELP

FILENAME:

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	inv_age<10	y	y	y	y	y	n	n									
C2	early_disc='N'	y	y	y	n	n	-										
C3	inv_total>100	y	n	y	-	-	-										
C4	inv_total<1000	y		n	y	n	y	n									
A1	write to hold_file	1															
A2	print pay_authorisatio		1		1		1										
A3	write to cash_request			1		1		1									

FIGURE 4 - Example Converted to Decision Table

The decision table can now be expanded to show the full set of condition columns - 2^4 (figure 5). If any additions, deletions, or changes to condition tests or actions were to be required, they would be done at this stage, with the columns of condition permutations expanding or contracting if new conditions are added or existing conditions are deleted.

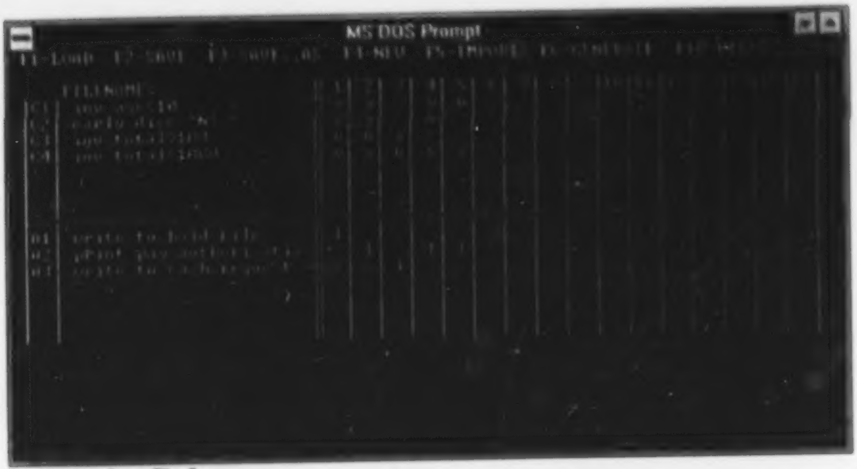


	FILENAME:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
C1	[illegible]	y	y	y	y	y	y	y	y	n	n	n	n	n	n	n	n
C2	[illegible]	y	y	y	y	n	n	n	n	y	y	y	y	n	n	n	n
C3	[illegible]	y	y	n	n	y	y	n	n	y	y	n	n	y	y	n	n
C4	[illegible]	y	n	y	n	y	n	y	n	y	n	y	n	y	n	y	n
A1	write to [illegible] file	1															
A2	print pay authorisation			1	1	1		1		1		1		1		1	
A3	write to cash request		1				1		1		1		1		1		1

Figure 5 - Full Decision Table

The tool or the user can then flag impossible combinations of conditions, such as columns 4, 8, 12 and 16 in figure 5 where clearly the inv_total cannot be < = 100 and > = 1000. These columns can either be deleted or an error action introduced. Here we will delete them.

Applying the first simplification rule already described, columns can be combined to produce figure 6.



Derivation from Fig 5:

Col 1 from 1; 2 from 3; 3 from (2+6)+(10+14); 4 from 5+7; 5 from (9+11)+(13+15).

FIGURE 6 - Simplified Decision Table

Code can then be generated from this table by rules such as looking for the exceptional 'Y' or 'N' in any row and using that as the first condition test, eliminating that row and column, then repeating on the remainder. Applying this principle to figure 6 gives figure 7 - a much more compact and understandable presentation of the logic than the original. Approaches to the generation of code from tables are discussed by Humby (1973) and Pollack (1965).

```
IF inv_total not < 1000
        THEN write to cash_request
ELSE    IF inv_age < 10 and early_disc = 'N' and inv_total > 100 write to hold file
        ELSE print pay_authorisation
```

Figure 7 - Optimised Code

The tool also gives the option to instead generate rule style code as illustrated by figure 8.

```
*************************************************************************
*----------------------- RULE 1 ------------------------
IF inv_age<10 AND early_disc='N' AND inv_total>100 AND inv_total<1000
        write_to_hold_file
ENDIF
*----------------------- RULE 2 ------------------------
IF inv_age<10 AND early_disc='N' AND NOT inv_total>100 AND inv_total<1000
        print_pay_authorisation
ENDIF
*----------------------- RULE 3 ------------------------
IF inv_total>100 AND NOT inv_total<1000
        write_to_cash_request
ENDIF
*----------------------- RULE 4 ------------------------
IF inv_age<10 AND NOT early_disc='N' AND inv_total<1000
        print_pay_authorisation
ENDIF
*----------------------- RULE 5 ------------------------
IF NOT inv_age<10 AND inv_total<1000
        print_pay_authorisation
ENDIF
*************************************************************************
```

Figure 8 - Rule based logic

4. FURTHER FACILITIES

The basic tool as described will be expanded to provide facilities such as:

Display and possible editing in the equivalent decision tree.

Processing the conditions and actions via a translation dictionary when going from code to decision table and vice versa, such that the decision table is easily comprehended - eg code 'If c-flag=1' becomes decision table condition 'Credit OK'. This has possibilities in converting code from one language to others, with the decision table as a common, language independent base.

Giving options of generating code for minimum storage or minimum execution time.

Processing more complex versions of decision tables - extended entry decision tables, and nested decision tables.

5. REFERENCES

Boggio-Togna, G.A., (1991), Scratchpad for truth-functional logic, *Computer Language*, Vol 8, 12 pp. 40-50.

Celko, J., (1983), A simplified algorithmic approach to decision tables, *Byte*, Vol 8, 11 pp. 507-514.

Coffee, P., (1991), Logic Gem clarifies program design, *PC Week*, Vol 8, 19 pp.112-113.

Computerworld, (1988), CASE tool kits workbenches, June 6, 1988.

Gane, C and Sarson, T., (1992), " Structured Systems Analysis, Tools and Techniques", Prentice-Hall, Englewood Cliffs, N.J.

Humby, E. (1973), "Programs from Decision Tables", American Elsevier Publishing.

Hurley, R.B., (1983), "Decision Tables in Software Engineering ", Van Nostrand Reinhold.

McCarthy, J., (1986), Decision tables and logic processing. *Computer Language*, Vol 3, 11, pp. 73-80.

Minasi, M.(1990), Decision tables: a form of expert system, *AI Expert*, Vol 5, 11 pp. 15-20.

Subramanian, G. et al, (1992), A comparison of the decision table and tree, *Communications of the ACM*, Vol 35, 1 pp. 89-94.

Vessey, I. and Weber, R, (1986), Structured tools and conditional logic: An empirical investigation, *Communications of the ACM*, Vol 29, 1 pp. 48-57.

Specific Aspects of IE

THE INFORMATION NETWORK SYSTEM

R.W. van der Pol, R.A.U. Quast

University of Limburg, Faculty of General Sciences,
P.O. Box 616, 6200 MD Maastricht, The Netherlands

Abstract

Today's information technology offers access to large amounts of documents containing text, image, audio and video. As a result of the huge amount of information finding relevant documents has become a difficult task. We assume that appropriate knowledge of relations amoung documents attributes to finding the relevant documents. This paper describes an Information Network System applied to portfolio management which contains such relations. The system presents these relations in a graphical manner.

1. RESEARCH GOAL

It is generally known that solving problems requires information. In many cases this information is stored in documents. Finding relevant documents for solving problems is subject of our research. With the development of new technologies, the amount of documents electronically available has grown enormously. Therefore, finding relevant documents requires more effort. We assume that appropriate knowledge of relations among documents is essential to our research. Traditional selection methods result in a set of documents which are most of the time loosely coupled. Having to our disposal appropriate knowledge, we have developed a system, called 'Information Network System' (INS) which represents document names and their mutual relations in a graphical manner. INS has access to the underlying multi-media documents.

In order to show that it is essential to have appropriate relations, we applied INS to portfolio management by incorporating INS in a system called 'Multi-Media Portfolio Management System' (MPMS). One of the components of MPMS is a database containing references to documents stored on hard disk and on CD-ROM. These documents consist of a combination of text, image, audio and video. All documents are relevant to portfolio management. However, for specific problems only a few of these documents are worth examining. INS supports the selection of these documents.

Our research goal is how to formulate the appropriate relations as prompted by the contents of the documents.

Section 2 focuses on INS. Section 3 gives some background knowledge to the field of portfolio management. The presentation of MPMS database is described in section 4.

2. INFORMATION NETWORK SYSTEM

INS consists of the following three components:

- the access component: for accessing information in databases;
- the selection component: for selecting information in databases;
- the presentation component: for presenting the information selected.

In figure 1, the relations between these components are illustrated, moreover the relations to MPMS (1). The access component (7) enables INS (2) to read via relation (8) the contentss of the database (9). The database contains document names with references (10) to multi-media documents (11) and their application programs (12). It also contains keywords and other features of the documents. Documents and their application programs are stored on hard disk and CD-ROM. The selection component (5) enables INS to select via relation (6a) document names on the basis of the document features. This component is used to make a first selection of documents. The presentation component (3) shows via relation (4) the document names and relations among them, both stored in the database. Because of its link (6b) to the access component, the presentation component is also able to show the contents of the documents.

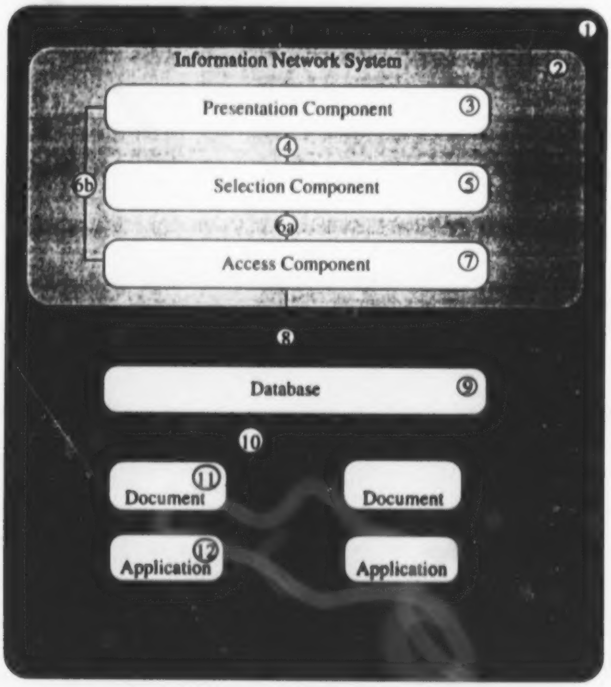

①
Information Network System
②
Presentation Component
③
④
6b
Selection Component
⑤
6a
Access Component
⑦
⑧
Database
⑨
⑩
⑪
Document
Document
⑫
Application
Application

Figure 1. Components of INS and MPMS

The selection component offers a Boolean query mechanism as described by Date (1990). Such a traditional search technique in many cases does not provide sufficient facilities for finding relevant documents. In order to find ways of improving these techniques, we studied the process of finding and using information in the field of portfolio management.

3. PORTFOLIO MANAGEMENT

Portfolio managers control assets in such a way that, given alternative investment opportunities with equal risk, returns are maximized (see Reilly, 1989). Controlling these assets requires appropriate knowledge of the following three categories of information:

- macro economic information: e.g., inflation, interest and money supply;
- company information: e.g., profits, solvency and liquidity;
- statistical information: e.g., trading volumes of stocks, specific stock prices and average stock prices.

This information to a large extent consists of documents in the form of news reports. Portfolio managers examine these documents and compare several documents to each other. MPMS supports the analyses of all three information categories. It uses the facilities of INS for the process of finding relevant documents. This process consists of the following four steps:

1. selecting documents in the document table (table 1);
2. selecting documents in the subject table (table 2);
3. selecting documents in the relation tables (table 3);
4. inspecting relevant documents.

The *document table* contains the following five columns: the name of the document, a reference to the actual document and its application program, the date the document has been issued, the category of information to which the documents belong, and the importance of the document. The entries in the date, category and importance columns serve the first selection step by allowing execution of queries on documents in these columns.

Document Name	Reference	Date	Category	Importance
Inflation_1	OLEobject[1]	1/1/93	Econ	0.6
IBM	OLEobject	1/2/93	Fund	0.7
Brasil	OLEobject	1/2/93	Econ	0.9

Table 1. Document Table.

---

[1] OLEobject (Object Linking and Embedding) are Windows' standard references to documents and their application programs, which are stored on different media in different forms. Documents have the form of text, image, audio and video. Examples of applications are word processors, databases and knowledge bases.

The second selection step filters documents addressing the same subject(s), also by executing queries.

Document Name	Subject
Inflation_1	Inflation
Inflation_2	Inflation
IBM_1	Balance Sheet

Table 2. Subject Table.

In the third step documents are selected by examining their mutual relations. Portfolio managers do not store the relations used for comparing documents. Storing these relations most likely would result in more effective selections. For this reason we created three *relation tables* (table 3). Table 3 shows for example, that the document with name 'IBM_2' is an update of the document with the name 'IBM_1'.

Document Name	Updates
IBM_2	IBM_1
Inflation_2	Inflation_1
Inflation_2	Japan_1

Document Name	Agrees
IBM_2	Japan_1
Inflation_2	Bundesbank_1
Inflation_2	Japan_1

Document Name	Contradicts
IBM_2	J_P_Morgan
Inflation_2	Brasil
Inflation_2	OESO_1

Table 3. Relation Tables.

When experimenting with MPMS, the extensive coherence among documents proved to be a serious obstacle for selecting relevant documents. In order to overcome this obstacle we have explored new ways of presenting the contents of the MPMS database.

4. PRESENTATION OF THE MPMS DATABASE

This chapter discusses how the contents of the MPMS database is presented, such that the coherence among documents becomes clear. The discussion is based upon an example of a task.

Suppose that the portfolio manager wants to perform one of his tasks: predicting the inflation. Using the query mechanism of the selection component, he performs steps 1 and 2 as described in the previous chapter: he selects the documents in the category 'economic' that have the subject 'inflation'. The result is presented by INS, sorted according to date (table 4).

With the relations among documents explicitly available in the table, the portfolio manager distinguishes between documents relevant for the prediction and documents irrelevant for the prediction, without having to read all the documents in the table.

document_name	date	imp.	agrees_with	contradicts	updates
Inflation_12	24/07/94	0.8	Second_Quarter		
			Investing_in_NL		
					Inflation_11
					Nat_Budget
Second_Quarter	07/07/94	0.9	Inflation_12		
				Investing_in_NL	
				Inflation_11	
					Europ_Exchange
Investing_in_NL	06/07/94	0.9	Inflation_12		
				Second_quarter	
British_C_Bank	25/06/94	0.9	Europ_Exchange		
			DNB_93		
			Inflation_8		
				Clinton_3	
Europ_Exchange	24/06/94	0.9	British_C_Bank		
				Inflation_11	

Table 4. Table presentation.

For certain purposes, presentation in a table format is not the best alternative. According to Coll (1994), specific values are quickest and most reliably obtained from tables, but relational information (i.e. relations between values) is better shown in graphs. The date, category and importance in the document table and the relations in the relation tables are considered values. The relations among values that are not explicitly present as values in one of the tables are considered relational information. Taking into account the findings of Coll, we expect a graphical presentation of the table to improve the process of distinguishing between relevant and irrelevant documents. For this reason, we developed facilities for a graphical presentation of the MPMS database: a two dimensional (2D) presentation and a three dimensional (3D) presentation. The 2D presentation was based on ideas put forward by Parsaye (1989). Both presentations are discussed hereafter.

The 2D presentation is started after selection of a document from table 4, e.g. the most recent document. It shows the document, its directly related documents and relations. Commands are available for expanding and reducing the obtained network and for presenting the contents of documents. After reading the contents, one switches back to the 2D presentation. Figure 2 shows an example of the 2D presentation. The importance and the date of issue are not visible, but can optionally be shown along with the document identification.

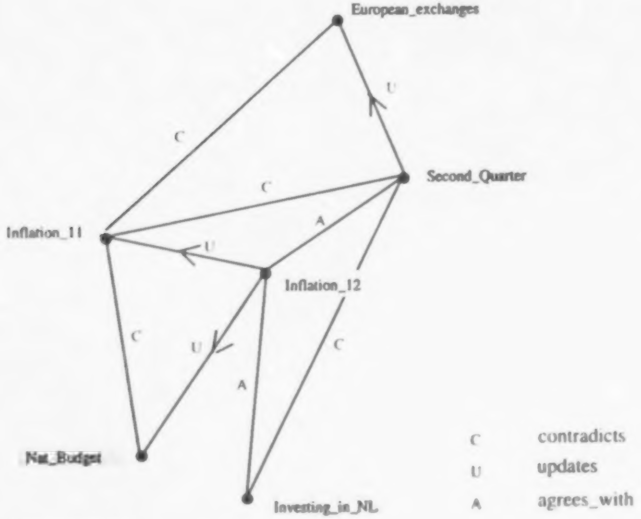

European_exchanges
U
C
C
Second_Quarter
A
Inflation_11
U
Inflation_12
C
U
C
A
Nat_Budget
Investing_in_NL
C contradicts
U updates
A agrees_with

Figure 2. 2D presentation.

In the 2D presentation the explicit relations among documents are clear. Conversely, the implicit relations between pairs of documents, as captured in their time of issue, are difficult to grasp. They are understood by the portfolio manager only after reading a number of dates. The same argument holds for the implicit importance relations between pairs of documents. In an attempt to improve the understanding of the implicit time and importance relations, we developed the 3D presentation shown in figure 3.

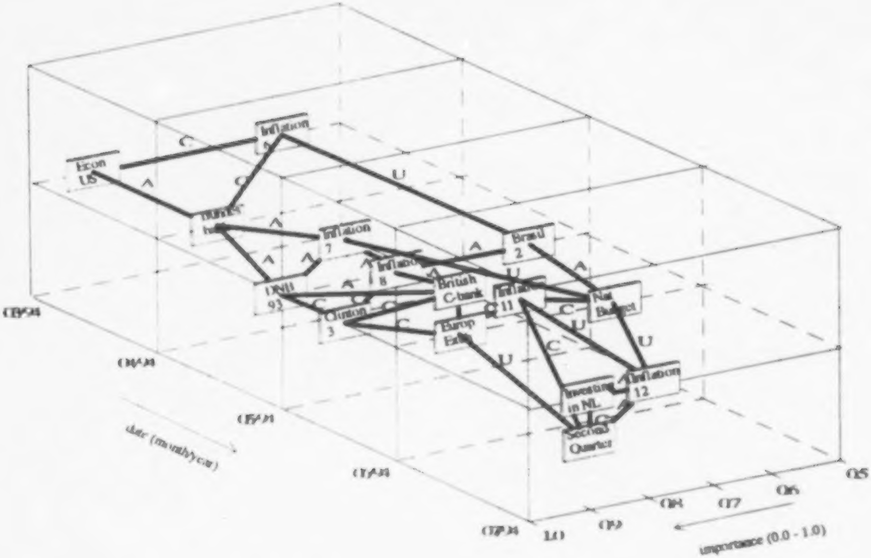

Econ US
Inflation 6
Inflation 7
Inflation 8
DNB 93
Clinton 3
British C-bank
Inflation 11
Europ Exch
Brazil 2
Nat Budget
Investing in NL
Inflation 12
Second Quarter
date (month/year)
importance (0.0 - 1.0)
1.0
0.9
0.8
0.7
0.6
0.5

Figure 3. 3D presentation

The 3D presentation is started by entering the desired time interval instead of selecting a document. It then shows documents issued in the time interval as identified boxes in a three dimensional space. The relations among the documents are shown as rods connecting the boxes. One axis of the three dimensional space measures time, another measures importance. The third axis has no scale, but accommodates for documents having identical date and importance. Without the third axis, such documents would coincide. The point of view can be altered in order to emphasize either the time or the importance aspect.

In addition to the implicit relations understood from time and importance, the 3D presentation gives insight in relations that have to be derived from the explicit relations. For example, it can easily be seen that the document 'Clinton_3' is a 'dissident' document (i.e. it has a high number of documents that contradict it, compared to the number of documents that agree with it). This is even clearer when the relations are indicated by different colors. Other instances of implicit relations, not shown in figure 3, are a sudden change of common opinion, and a period of increased writing activity. Both events are usually relevant for valuing the contents of documents.

In a full scale application of MPMS the number of documents and relations may be excessive and thus reduce the advantages of the graphical presentations. If that situation occurs, the number of documents and relations is decreased by activating filters that suppress the visibility of specific documents and relations.

5. CONCLUSIONS

We experienced that, by introducing relations among documents, the selection process of relevant documents for the prediction of economic variables improved. By presenting the relations graphically, a second improvement in the process was obtained.

One of our current research goals is integrating knowledge bases in INS for application in the field of portfolio management. The resulting system should provide facilities for inspecting knowledge bases about selected subjects.

6. REFERENCES

Date, C.J (1990), "An Introduction to Database Systems, Volume I", Addison-Wesley Publishing Company, Readings, Massachusetts.

Coll, R.A., Coll, J.H., Thakur, G., (1994), Graphs and tables, a four-factor experiment. *Communications of the ACM* April 1994, Vol. 37, no.4, pp. 78 87.

Parsaye, K., Chignell, M., Khoshafian S. and Wong, H. (1989), "Intelligent Databases; Object-Oriented, Deductive Hypermedia Technologies", John Wiley and Sons, New York.

Reilly, F.K. (1987), "Investment Analysis and Portfolio Management", The Dryden Press, Forth Worth

SECURITY ASSESSMENT DURING INFORMATION SYSTEMS DEVELOPMENT

Alenka Hudoklin, Branislav Šmitek

University of Maribor, School of Organizational Sciences, 64000 Kranj, Prešernova 11, Slovenia

Abstract

A security assessment method based on the identification of relevant chains "threat source - threat - exposure - asset" is presented. It is useful in early stages of information systems development when all data needed for conventional risk analysis are not yet available.

1. INTRODUCTION

It is well known that information technology comprising computers, electronic data networks and other ancillary facilities and services bring considerable economic and social benefits to organizations. On the other hand, organizations are critically dependent on information system resources. Failures or misuse of these resources could put organization out of business. Organization vulnerability to security threats begun to be acknowledged by management only in recent years. Nevertheless, in most organizations, except the larger ones, the countermeasures to information system security threats are still inadequate.

Security issues must be considered during all stages of information system development. In this report a model of security assessment during the development of an information system is presented.

2. BASIC TERMS

Terms used in computer security analysis are not always clear nor adequately defined. The same word is used for different things or different words are used for the same thing. The recently published Dictionary of concepts, standards and terms (Longley, 1992) is of great help but still definitions of basic terms are not unique. In the framework of this contribution we will try to define precisely terms used in the following text.

Security is the ability of a system to perform required functions in specified conditions without accidental events or events caused by unintentional or intentional human activity which can disrupt system integrity/normal activity.

Asset is a system unit which can be treated independently in security analysis.

Security threat is an event or activity with the potential to cause damage to a system/asset or operation.

Security threat source is an event or activity which causes one or more different security threats.

Exposure of the system/asset is the result of a process caused by the realization of a specific security threat.

Risk is the product of the probability/frequency of a realized threat and expected loss caused by the threat.

Countermeasure is a direction or activity which prevents/detects occurrence of a security threat or repairs the damage caused by occurrence of a security threat.

Protection is the assurance of system security by means of countermeasures and activities preventing disruption of system integrity/normal activity.

3. SECURITY ASSESSMENT

3. 1 Traditional approach

Risk analysis is the most important step in information system security assessment. Risk analysis procedures are usually based on traditional model shown in Figure 1.

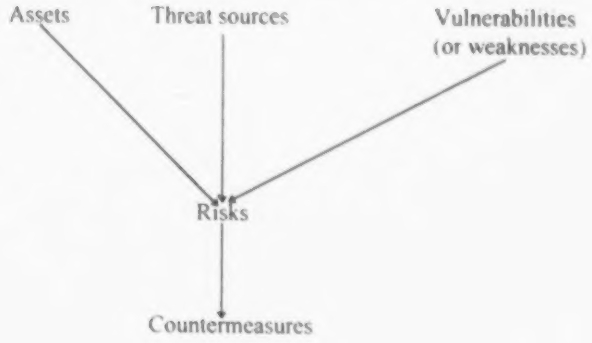


Figure 1. Risk analysis.

Conventional risk analysis involves identification and evaluation of assets, threats and vulnerabilities which leads to risk assessment and, finally, to determination of appropriate countermeasures.

On the basis of this model numerous quantitative and qualitative methods of risk analysis have been developed (Hudoklin 1993). Some of them are implemented as software products (Wong, 1990). Risk analysis requires considerable amount of data on system assets and security threats as well as expert knowledge. During the development of an information system all data needed for evaluation procedures are not yet available. It is possibly one of the reasons that all to often security is not considered until the information system is well into the implementation stage. However, the security features should be incorporated into the information system during the earliest design stages. A systematical approach to assess the nature and impact of security threats would be useful (Ettinger, 1993).

3.2 Assessment during information system development

The essential part of security assessment is identification of assets and threats, finding out the threat sources and assets exposures and defining possible chains threat source - threat - exposure - asset. The number of this chains in a real information system is very large. It depends on information system size and location. We tried to reduce the number of possible chains threat-threat source-exposure-asset by grouping individual chain elements into following categories:

- external resources
- infrastructure
- hardware and communication equipment
- software
- data
- human

Having in mind that the aim of risk analysis is to find appropriate countermeasures, security threats and threat sources must be separately identified.

Security threats were grouped into the following categories:

- fire, rising/falling water or other fluids
- asset failure
- intrusion, theft, blackmail, espionage, sabotage

The next step consists of finding possible threat sources and exposures of assets for each group of threats.

Possible threat sources were grouped into 4 classes:

- environmental factors
- technical or biological failure
- unintentional human activity
- intentional human activity

Exposures of assets were classified as follows:

- destruction, interruption, unavailability
- modification
- unauthorized disclosure, misuse

The groups of threat sources and assets were related through possible exposures as shown in Table 1:

Table 1. Assets exposures related to different threats sources.

Assets Threats source	External resources	Infra-structure	Hardware and communication equipment	Software	Data	Human
Environmental factors	**D M**	**D M**	**D M**	**D M**	–	**D M**
Technical or biological failure	**D M**	**D M**	**D M**	**D M**	–	**D M**
Unintentional human activity	**D M**	**D M**	**D M**	**D M**	**D M**	–
Intentional human activity	**D M U**	**D M U**	**D M U**	**D M U**	**D M U**	**D M**

Key:

D = destruction, interruption, unavailability
M = modification
U = unauthorized disclosure, misuse

Interrelations of defined groups of threat sources, threats, exposures and assets are shown in Figure 2.

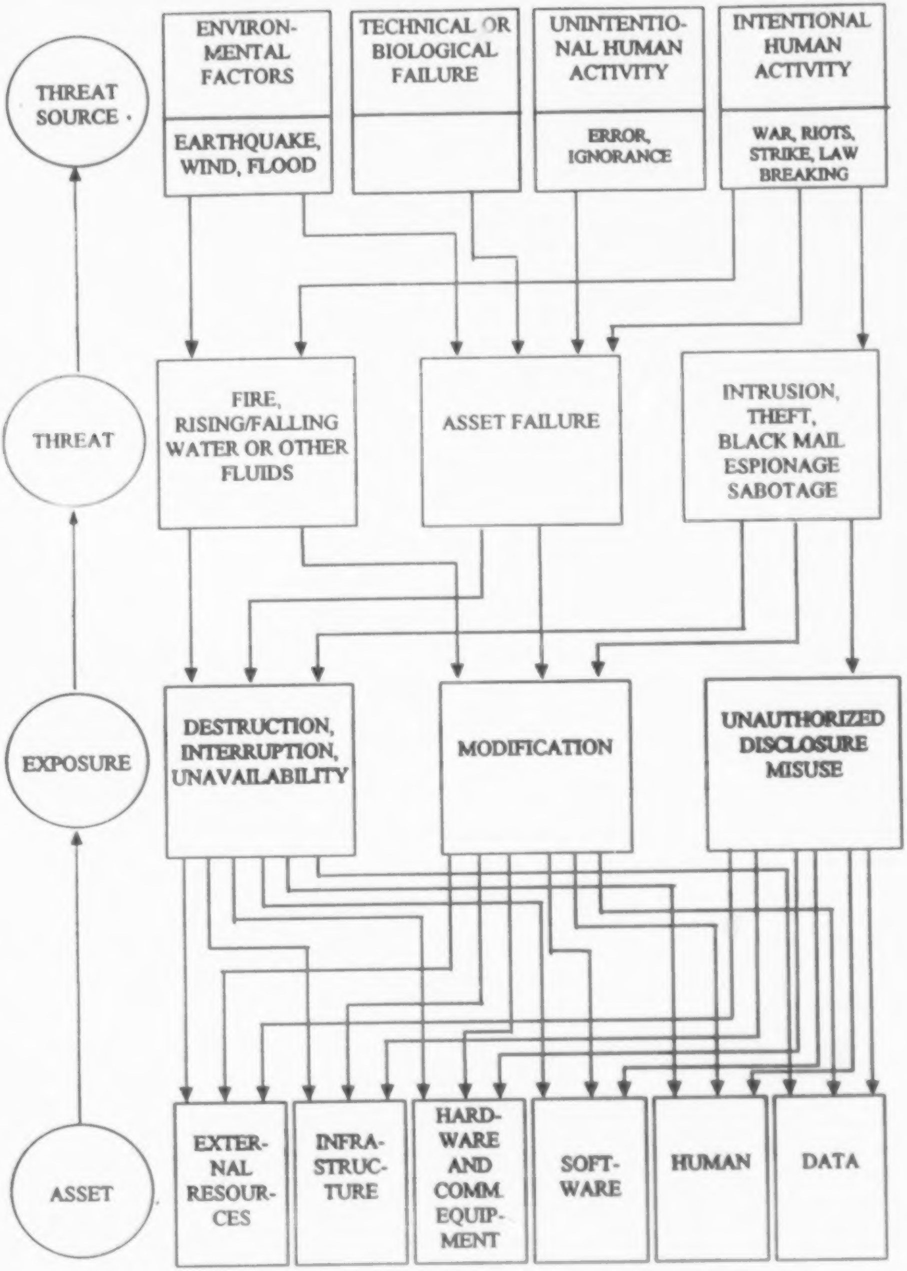

THREAT SOURCE
ENVIRON-MENTAL FACTORS
EARTHQUAKE, WIND, FLOOD
TECHNICAL OR BIOLOGICAL FAILURE
UNINTENTIONAL HUMAN ACTIVITY
ERROR, IGNORANCE
INTENTIONAL HUMAN ACTIVITY
WAR, RIOTS, STRIKE, LAW BREAKING
THREAT
FIRE, RISING/FALLING WATER OR OTHER FLUIDS
ASSET FAILURE
INTRUSION, THEFT, BLACK MAIL ESPIONAGE SABOTAGE
EXPOSURE
DESTRUCTION, INTERRUPTION, UNAVAILABILITY
MODIFICATION
UNAUTHORIZED DISCLOSURE MISUSE
ASSET
EXTER-NAL RESOUR-CES
INFRA-STRUC-TURE
HARD-WARE AND COMM. EQUIP-MENT
SOFT-WARE
HUMAN
DATA

Figure 2. Relationship between threat sources, threats, exposures and assets

Each asset being identified during the development of an information system can be included into one of six cited groups. Figure 2 enables to determine relevant exposure, threat and threat source categories related to a given asset group. Relationship between assets and threat source helps to find out a list of countermeasures providing protection to assets. However, the determination of the most effective countermeasure requires additional evaluation of security risks associated with a given chain "threat source - threat - exposure - asset".

4. CONCLUSION

During information system development security assessment based on traditional model of risk analysis could not be performed due to lack of needed data. On the other hand, basic security protection measures must be planned. The method described enables planning of security countermeasures in early stages of information system development without a detailed evaluation of security risks.

REFERENCES

Bhaskar, K, (1993), "Computer Security", NCC Blackwell Ltd, Oxford.

Ettinger J.E. (1993), Key issues in information security, in: "Information Security", Chapman & Hall, London.

Hudoklin A., Šmitek B., (1993), Kvantitativno vrednotenje tveganja groženj varnosti računalniško podprtega informacijskega sistema, Organizacija in kadri, Kranj, Vol. 26, pp. 180-184.

Longley D., Shain M., Caelli W.,(1992), "Information Security Dictionary of Concepts, Standards and Terms", Stockton Press, New York.

Šmitek B., Hudoklin A., (1993), Vgrajevanje varnosti v računalniško podprt informacijski sistem, Organizacija in kadri, Kranj, Vol. 26, pp. 185-191.

Wong K., Watt S., (1990), "Managing Information Security", Elsevier Science Publishers, pp. 179-181.

INFORMATION AS A FACTOR OF PRODUCTION: A QUESTION OF SYSTEMS DEFINITION

Bernd Schiemenz

Philipps-Universität Marburg, Fachbereich Wirtschaftswissenschaften, BWL I: Allg. Betriebswirtschaftslehre und Industriebetriebslehre, Am Plan 2, 35032 Marburg, Germany

Abstract

In economics and economy factors of production are an important concept. They focus on the question of preconditions and causes of production and wealth. The notions "information age" and "information pollution" show however, that information in this regard may be positive or negative. Therefore the attribution of the property of a factor of production to information and communication technique resp. systems, which is frequently found, seems not to meet the core of the problem. The deeper going question is, how far information itself, handled by this technique, is a factor of production. The paper shows results of the actual discussion of this question for information producing enterprises and gives an extension for enterprises producing material goods.

1. INTRODUCTION

From september 28th to october 2nd 1992, the 22nd congress of the German Informatics Society took place in Karlsruhe under the motto "Information as a Factor of Production" (Görke et al. 1992).

The papers are partly very interesting and by authors of a high reputation. From the point of view of a business economist however, who is acquainted with production factors, production functions and production theory, their topic is approximately "Information Systems in Production" and not "Information as a Factor of Production".

Only the first paper "Information as a Factor of Production", written by Prof. Dr.-Ing. Hartmut Weule, member of the board of directors and responsible for research and technique of the Daimler-Benz AG., gives a short explanation of production factors and information. But the content of the paper is very generally about (economic aspects of) information processing, too.

On the other hand, it seems more and more important to reflect not only the economic aspects of information processing - which is frequently beeing done under the topic of information management - but to get a deeper insight, what information really means, and if it is really a factor of production, without which the result of production processes cannot be explained. The following is an attempt into this direction on the basis of production theory.

2. INFORMATION

It is interesting, that within computer science a basic notion like information remains so little reflected. In the German encyclopedia "Duden Informatik" (1988, p. 273) we e.g. read: "For informatics as the science of the systematic processing of information the notion of 'information' should be of central importance; however, until now it has hardly been made precise."

The problem arises from the fact, that the term "information" is used for the different semiotic levels syntax, semantics and pragmatics. In the (signal-statistical) "information theory" one asks for the lowest number of bits to transmit or to store information. There, the term information is used on the syntactical level which is concerned with building words from the signs of an alphabet (of any nature).

In the context of "information system" or "information management" the term information is generally used on the level of semantics. Information documents the real or imagined world, but without relation to possible actions.

We finally have the level of pragmatics. Information there is used in the sense of purpose-oriented knowledge, where the purpose lies in the preparation of action (Wittmann 1969). This is the dominating comprehension in German business economics.

The difference shall be explained by the following sentence: "Metallgesellschaft Ag lost 2 billion DM in 1993".

On the syntactical level one may seek efficient codes in order to store or transmit this sentence with the lowest number of bits. To store the two letters DM one generally needs two bytes resp. 16 bits, though, by optimum coding, with 16 bits one could distinguish between 65536 objects, e.g. currencies. For optimum coding, 1 byte or 8 bits might suffice.

On the semantical level the sentence is one of many messages, one could read in the newspapers in January 1994. It says something about the economic situation of a specific german firm.

Whether the sentence is also an information on the pragmatical level depends on the specific situation of the reader. If all his assets are invested in stocks of Metallgesellschaft Ag the information content is very high. For me, the message was interesting but not able to influence my actions, and the pragmatical information content was nearly zero.

3. PRODUCTION THEORY AND FACTORS OF PRODUCTION

The production theory contains explicative systems about how nature (quality) and quantity of products (output) in time depend on the nature, quantity, input intensity and input time of the factors needed to produce these products. Such factors of production are for example materials, services, labour and production equipment (Busse von Colbe and Lassmann 1988, p. 71).

The production theory thus wants to give a quantitative relation between the quantities of one or several output goods and the several input goods of a production process. The output goods are called products. The input goods are called factors of production.

More formally one can say:
be x_l the quantity produced of the product X_l, $l = 1, 2, \ldots, s$
be r_i the quantity needed of the input good R_i, $i = 1, 2, \ldots, n$ -

then the intention of production theory is, to find production functions in one of the two typical forms:

explicit for the quantity of output as product-function:

$$x_l = f_l\,(r_1, r_2, \ldots, r_i, \ldots, r_n), \quad l = 1, 2, \ldots, s$$

or explicit for the quantity of input as productor (resp. factor of production or factor) function

$$r_i = g_i\,(x_1, x_2, \ldots, x_l, \ldots, x_s), \quad i = 1, 2, \ldots, n$$

Consequently, factors of production then are those material or immaterial goods, without which production functions cannot be built.

In the economic literature we find several classifications of these factors. In German business economics the dominating classification goes back to E. Gutenberg (1969). He distinguishes the three "elementary factors" (Elementarfaktoren) menschliche Arbeitsleistung (human labour), Werkstoffe (material) and Betriebsmittel (production means: machines, apparatusses, buildings and the like). From the elementary factor human labour he extracts, in a second step, the dispositive human labour, leaving as rest the object-related human labour (objektbezogene menschliche Arbeitsleistung). While the latter enters the product directly (typically manual labour, transforming the material) the former combines the rest of the factors. In other words: they plan, lead, organise and control the combination of object-related labour, material and production means.

The dispositive factor is to be seen functionally. Also the labour of a worker at his machine includes this factor in so far, as he has degrees of freedom to influence the production process. However, the relation between dispositive and object-related labour decreases from the top to the basis of the business hierarchy.

While material is beeing consumed during a production process (one therefore speaks of "Verbrauchsfaktoren": consumption factors), the rest of the factors (cum granu salis) refers only to a use of potentials ("Potentialfaktoren").

This traditional system of factors of production concentrates on material production processes. In recent years it has been expanded to include additional factors like services of other firms and use of environment. Material has been expanded to "object factors" (Objektfaktoren), in order to include also other objects on which production processes work, e. g. people and their knowledge as objects of educational institutions. Nominal factors (like cash) had been included without which especially a bank or an insurance company cannot work. And frequently we also find information as a seperate factor of production. Fig. 1 shows a rather general system of factors of production, elaborated by Bode (1992, here p. 80) on the basis of earlier publications of Kern and Fallaschinski (1978/1979).

1. Human labour
1.1 dispositive labour (management)
    planning
    organisation
    control
    direction (executive board)
1.2 object related labour
    mental
    physical

2. Production means (in the broader sense)
2.1 for use (production means in the narrower sense)
    passive ones
    - of material type (e.g. land, buildings)
    - of immaterial type (e.g attainments, knowledge, rights)
    active ones (e.g. machines, kettles, EDP-systems)
2.2 for consumption (supplies incl. energy)

3. Objectfactors
3.1 internal objectfactors (e.g. raw material)
3.2 external objectfactors (e.g. tangible assets, persons, attainments)
3.3 objects running through

4. Supplementary factors
4.1 direct services of others
4.2 indirect support services
4.3 use of environment

5. Nominalfactors (financial resources as cash liquidity)
5.1 for use as means of payment (e.g. in banks)
5.2 for insurance benefits (liquidity)
5.3 for provision of security

6. Information

Fig. 1: A system of factors of production (Bode 1992, here p. 80, on the basis of Kern and Fallaschinski 1978 and 1979)

4. INFORMATION AS A FACTOR OF PRODUCTION IN INFORMATIONAL PROCESSES

The book of Bode (1992) intends a production-theoretical analysis of the production of information of a firm. As a preparation for this he analyses the relation between the factors of production, shown in fig. 1, and information and comes to the conclusion "...that only in rare cases a classification possibility is given that is unequivocal and generally valid. Information can play the role of an object factor, a production means, a supplementary factor, or it can finally belong to a separate own category of factors. It is an input and essential output of dispositive decision processes and possesses properties that belong as well to such factors which are used as to those which are consumed. They can appear as external or internal factors. ...The fact that some factors can be put in order in several ways can be reduced when the general validity is abandoned and the respective specific purpose of the classification is clearly indicated. If this purpose refers to the infor-

1. Human labour
1.1 dispositive labour (management)
    planning^{+}
    organisation^{+}
    control^{+}
    direction (executive board)
1.2 object related labour
    mental^{+}
    physical

2. Production means (in the broader sense)
2.1 for use (production means in the narrower sense)
    PASSIVE ONES *(e.g. recipes, process instructions, algorithms)*
    active ones (e.g. machines, kettles, EDP-systems)
2.2 for consumption (supplies incl. energy)

3. Objectfactors
3.1 INTERNAL OBJECTFACTORS *(e.g. cost information for internal accounting)*
3.2 EXTERNAL OBJECTFACTORS *(e.g.customer specific information to make the balance sheet for tax purposes by a tax consultant)*
3.3 OBJECTS RUNNING THROUGH *(information as an object of trade, e. g. standard software, books)*

4. Supplementary factors
4.1 DIRECT SERVICES OF OTHERS *(e.g. consulting services)*
4.2 indirect support services
4.3 use of environment

5. Nominalfactors (financial resources as cash liquidity)
5.1 for use as means of payment (e.g. of banks)
5.2 for insurance benefits (liquidity)
5.3 for provision of security

6. Information *(residual category, e. g. information output of dispositive planning processes)*

Fig. 2: Information in the system of factors of production (Bode 1992, p. 96)

mation system of the enterprise, it is well justified and reasonable to bring in an own factor category *information*. In all other cases a major part of information can be included into other factor categories, depending on their role in the production process. Solely information as an input and output of dispositive decision processes remains - independent of the purpose of the investigation - an element of the factor category *information*, because they can neither be seen as object factors nor as production means or supplementary factor." (Bode 1992, p. 95)

Fig. 2, which summarizes the result of his analysis, shows information in the system of factors of production. Categories of factors, which include information, are printed in capital letters. In brackets and italics corresponding examples are added. A cross indicates factors, where information is the output of dispositive planning, organising and control processes as well as of mental workprocesses.

5. SOME RECURSIVE AND HIERARCHICAL RELATIONS BETWEEN MATERIAL AND INFORMATIONAL PRODUCTION PROCESSES

At least in German business economics a systematic investigation of "information" as a factor of production in production processes with others than informational outputs is still lacking. (Bode 1992, p. 79)

The following part of the paper wants to give some ideas into this direction.

In order to start from a concrete basis we consider a flexible manufacturing system (FMS) (Schiemenz 1980, p. 54 ff.). This is a production system containing generally several and different machines, connected by transport installations moving the material and intermediate products from machine to machine. The third unit is a control unit. It extracts signals from the input, the production system and the output, transmits, stores and processes this information and uses the results to steer the process. The purpose of such a FMS is to transform material as input into final or intermediate products as output. It is nearly automated, so that there is little dispositive or object related human labour.

In this process we obviously find information and, indeed, information on the three semiotic levels of syntax, semantics and pragmatics. If the flexible manufacturing system is a sophisticated one, the informational process is directed towards purposeful knowledge, i.e. information on the pragmatic level. The information serves the purpose to make the production process possible or even to optimize this process.

But how far can we now speak of information as a factor of production of its own?

If we look at the flexible manufacturing system as a whole, the informational process is quasi built into this system. Input of the system is, as long as it works fully automated, only raw materials. Output are the products.

On this level of abstraction or system level the situation is isomorphic to that of a traditional machine for which E. Gutenberg (1969) developed his production function type B on the basis of consumption functions . These consumption functions are specific for a machine (or a machine system) j and relate (in the case of an adaptation to a changing work load by changing intensity) the amount r_{ij} of factor i used by this machine to the intensity d, with which the machine works as well as to the relevant parameters z_{1j}, z_{2j}, ...z_{vj} of the machine. The z_{ij}, $i = 1, 2, \ldots, v$, together build the so called z-situation.

$$r_{ij} = f_{ij}(z_{1j}, z_{2j}, \ldots\ldots z_{vj}; d_j)$$

The intensity (d) of the machine itself depends on the output x, one requires from the machine, finally the quantity of the product. We therefore get a production function (for one factor of production R_i and one machine j) of the type

$$r_{ij} = f_{ij}(z_{1j}, z_{2j}, \ldots\ldots z_{vj}; d_j(x))$$

In this production function we find no information as factor of production. The specific way of collecting, transmitting, storing and processing information and using it for activating purposes is quasi included, we could also say modelled, in the z-situation.

The situation is different when we separate our flexible manufacturing system into a material processing system and an information processing system. Parts of the information processing system are (at least) transducers (they give signals about the material process), transmission units, storing units, processing units (in the narrower sense of a CPU) and final control elements (they transform information into changes in the material processing system). The flowing or stored signals (measured or for activation) themselves may be seen as part of the information processing system as well. The other units of the flexible manufacturing system (machines, transmission resp. transport elements) constitute the material processing system. The transducers and the final control elements are situated at the boundary between the two subsystems.

If we try to build a production function for this material processing system alone, material factors would not suffice. Indeed, this unit (without controller) would produce nothing, even if we (tried to) input the same material as before. If, however, some "demon" would provide us with the correct signals for the activation and we would input them into the final control elements, we would get output, in the case of the same material and the same information the same output as from the unseparated system.

Let us look at this demon. It could have the structure of a feed forward controller or of a feed back controller. Combinations of both are possible, too. The feed forward controller may get input-information (e.g. parameter values of the material) or not. In both cases it must contain the necessary knowledge of the controlled material process which enables it, to expedit the right information at the right time (Schiemenz 1982, p. 27 ff.). This knowledge has the property of an informational potential factor. As long as the material system remains the same, there is no consumption of this potential factor, only its use. Knowledge needs a material bearer of information (chips, wires etc.), but its essence is informational.

This essentially informational potential factor has an interesting property: It can be copied and distributed nearly without cost as Noble laureate H. A. Simon (1967) pointed out in an early article. The information itself, which is provided, is consumed in the production process and may be seen as a consumption factor (like material).

Such feed forward controllers are, in general, not very efficient and effective. As soon as something happens in the material process, which is not already incorporated in the controller, the result differs from what the controller intends. And unexpected or unpredictible things do happen in nearly any process. Therefore, good controllers use also data about the state and the output of the material process to calculate the activating signals. These data then are information, - purposeful knowledge, - the (feedback-) controller needs as input to derive the correct activating signals.

Is this feedback-information also a factor of production for the flexible manufacturing system? The answer depends on what we include into the system. If we would exclude the connection between the transducers and the controller from the defined system, the feedback information would obviously be a factor of production. If one would cut this connection, the output would (at least in general) be less than with a functioning connection. On the other hand: If we would buy the feedback information from another demon and input it into the cut connection, the result would be the same as for an uncutted connection.

As a result of our investigation we can state that it depends on the definition of the (here only as an example flexible manufacturing) system, if the property of a factor of production must be attributed to information.

Similar to management control, we can identify different hierarchical levels of the productive factor information. Before we attributed the property of a potential factor to the demon, providing us with feed forward information. If there is a "meta-demon" providing us with the mentioned demon, we must attribute the property of factor of production to it as well, etc.

We find a similar pattern if we go into the details of our FMS. The parts of the material processing system (e.g. one of the machines) may itself be further divided into an information processing system and a material processing system, with similar conse-

483

quences as for the FMS, but now one system level lower. In other words: We find recursive relations (see Schiemenz 1994) of the form

material process := informational process + material process

All information needs a physical carrier. A letter, e. g. is in essence of an informational nature. The transport of the letter is however in essence a material one. We therefore frequently also find recursive relations of the form

informational process := material process + informational process.

The material process of an informational process itself may again contain informational elements, e.g. the control of the transport of a letter. Etc.

6. DISPOSITIVE INFORMATION AS A DISPOSITIVE FACTOR AND AS A PRODUCTION MEANS

In German business economics the dispositive factor is derived from the (elementary) factor human labour. The dispositive factor is therefore seen as of human nature, too.

Then the problem arises, how the informational subprocess controlling a material subprocess should be called. If the informational subprocess is realized by a person, it is called dispositive factor. If the function is programmed and realized by a process computer, the function within the process remains the same. From a functional point of view, we then should call it ,dispositive factor, too. A further reason is, that the core of the comprehension "dispositive factor" is its combinative function, this means, its capability to determine, which factors should be combined with which others and in what quantity.

My suggestion therefore is, to *distinguish two types of dispositive factors* according to the nature of its "carrier": *Human dispositive factor and artificial dispositive factor.* In the same way as the human dispositive factor is beeing derived from the (elementary) factor human labour, the artificial dispositive factor is derived from the (elementary) factor production means. It is no contradiction, that the artificial dispositive factor once had to be produced (e.g. programmed) by the human factor, because just the same holds for material production means.

7. SOME CONSIDERATIONS TOWARDS AN INFORMATION ORIENTED PRODUCTION THEORY

Informational processes are of increasing importance. One reason is the general trend towards an information-society. This trend is interrelated with a second trend, the development and utilization of information and communication mechanisms. It therefore appears sensible, to include this development also into production theoretical considerations.

In information production - e. g. that of an information broker - information as an object factor plays an important role. There it will frequently be possible, to build quantitative information production functions on a semantical level (see Bode 1992).

For informational subprocesses of material processes the situation will be much more difficult. But also here, the information may be partly quantifiable on the semantical level. In stock-control, a subproblem of material processes, e. g. the average stock level can be reduced, when the control frequency is beeing increased. Increase of control frequency, however, is identical with the increase of the input of information about the actual stock level.

Very general is also the possibility of an activity analysis. In activity analysis one describes a production by a list (a set or a vector) of figures, which indicate the quantities of the different (input and output) goods, involved in the production (see Wittmann 1993, here col. 3494).

For a FMS such lists would contain an exact description of the material and the informational subsystems, the different materials, human labour etc. and their quantities, necessery to produce a specific output programme. The subsystems could be described in more detail. For the informational subsystem this would for example include a description of the specific hard- and software used. Hard- and software themselves could be further detailed by indicating their single elements. For software this could imply a listing of the code, because already different codes of principally the same algorithm could produce different results.

If one has different activities leading to the same output of goods and if one knows the prices of the input goods, one could calculate the overall cost of the input and select the activity with minimum cost. Fig 3 shows this for two lumped input factors "material equipment" and "informational equipment" and one (output-)product. For a specific production process the production functions will be limitational and linear. Limitational means that the two input factors must be in a determined relation. If we keep the quantity of one factor constant and increase the quantity of the other factor, there will be no increase of the output (therefore the rectangular lines of the same product quantities). To produce in a FMS a specific quantity x of output, we need the material subprocess a determined time t_m and we need the informational subsystem another determined time t_i. If we want to produce 2x, we need the double times 2 t_m and 2 t_i. Therefore the function is linear.

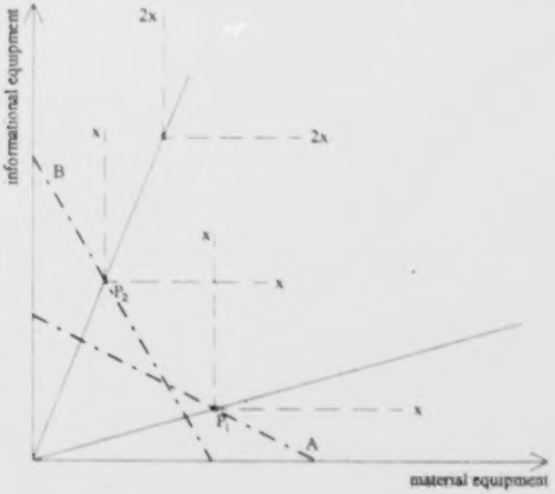


Fig. 3 Simple aggregated limitational linear production functions

Different material and informational equipment means a different process (e.g. a FMS of an other seller or only with another controller or only with another software etc.). As they differ, the two axises in fig. 3 must be understood rather abstract, somehow in the sense of "relative intensity of the material subprocess" and "relative intensity of the informational subprocess". The output quantities of the two activity points P_1 and P_2 in figure 3 are the same. In the process with P_1 the material subprocess is of higher importance, in the process with P_2 the informational subprocess is of higher importance.

Which of the two processes cause minimum cost depends on the cost relations between material and informational equipment. If material processes are relatively cheap, we would get a "iso-cost-line", a line of equal cost, of the type A. We would produce in point P_1 with the corresponding process. If informational processes are relatively cheap, we would get an iso-cost-line of type B and produce in point P_2.

One general tendency is, that the cost of (automated) informational processes per unit of performance decrease more than the cost of material processes. The reason is the increase in the productivity of information and communication equipment by approximately 50% per year over the last 30 years. One general result of our investigation therefore is, that the relative importance of informational subprocesses increased during the last years and will also increase in future. And that not only in our concrete example of a FMS, but also in more complex production systems (even as national or world economy) or in smaller processes as that of a single machine. But the statement could also be expanded to problems of transport, education etc. It is certainly a good strategy for a business firm to search for informational subsets in its activities and to try to increase their importance and finally to automate them. And it may be an even better strategy for a firm to look for new informational products to produce.

8. REFERENCES

Bode, Jurgen, (1992), "Die Produktion von Information - Eine produktionstheoretische Analyse der betrieblichen Informationserzeugung", wirtschaftswiss. Dissertation, Köln.

Busse von Colbe, Walther, and Lassmann, Gert, (1988), "Betriebswirtschaftstheorie", Bd. 1, 4. Aufl., Berlin et al.

Duden "Informatik", (1988), Mannheim et al.

Gorke, Winfried, Rininsland, Hermann, Syrbe, Max (Eds.), (1992), "Information als Produktionsfaktor", 22. GI-Jahrestagung, Berlin et al.

Gutenberg, Erich, (1969), "Grundlagen der Betriebswirtschaftslehre, 1. Bd.: Die Produktion", Berlin et al., 15. Aufl.

Kern, Werner, Fallaschinski, Karl-Heinz, (1978 and 1979), "Betriebswirtschaftliche Produktionsfaktoren (I und II)", in: WISU 7 (1978), p. 580 - 584 (I); WISU 8 (1979), p. 15 - 18 (II).

Schiemenz, Bernd, (1980), "Automatisierung der Produktion - Technische Voraussetzungen, verfahrensmäßige Erfordernisse und betriebswirtschaftliche Konsequenzen", Göttingen.

Schiemenz, Bernd, (1982), "Betriebskybernetik - Aspekte des betrieblichen Managements", Stuttgart.

Schiemenz, Bernd, (1994), "Hierarchie und Rekursion im nationalen und internationalen Management von Produktion und Information", in: "Internationales Management - Beiträge zur Zusammenarbeit", Schiemenz, Bernd, and Wurl, Hans-Jürgen, ed., Wiesbaden, p. 285 - 305.

Simon, Herbert A., (1967), "Programs as factors of production", in: California Management Review, Vol. X, No 2, 1967, pp. 15 - 22.

Wittmann, Waldemar, (1969), "Information", in: Handwörterbuch der Organisation, Stuttgart, Sp. 699 - 707.

Wittmann, Waldemar, (1993), "Produktionstheorie", in: Handwörterbuch der Betriebswirtschaft, 5. Aufl., Stuttgart, Sp. 3491 - 3518.

Quality

EXPERIENCES FROM AN INFORMATION RESOURCES QUALITY ASSURANCE PROGRAM THAT WORKED WELL, BUT ULTIMATELY FAILED

William A. Ohle[1] and Margaret Ann Bartlett[2]

[1] Ohle & Associates, 6005 Conway Road, Bethesda, MD 20817, USA
[2] The Compucare Company, 12110 Sunset Hills Road, Reston, VA 22090, USA

Abstract

The quality assurance function seems to operate in a cyclical fashion in most organizations. There is an initial struggle to get the function accepted as critical by senior management. The next phase, which may take the better part of a year, is to build a credible function that achieves results. The function operates for another year and then a change occurs which causes the organization to fall out of favor. Typically results are not achieved quickly enough, or new senior management enters that does not support the quality assurance function. Quality assurance is a necessary part of the system development process. The ultimate goal of any quality assurance function is to develop such consistent, successful processes and results that, if executed, will result in quality products. However, given the short cycle of 2 to 3 years that most quality functions survive, this is difficult to accomplish. The quality assurance function described in this paper fell into the same cycle.

1. INTRODUCTION

For a period of about 2½ years beginning in 1991, the authors had the rewarding and frustrating experience of building a proactive Quality Assurance (QA) program for the development of automated information systems. Our work was for a medium size United States of America (USA) government agency. The program included:

- Clearly defined measurement criteria;
- Assessment of business risks based on the criteria;
- Recommended actions to reduce levels of risks, and
- Oversight by senior agency executives

The purpose of the program was to provide useful guidance to applications developers and to ensure that senior executives had sufficient information upon which to base business decisions. The program succeeded in a business sense, but ultimately collapsed under political pressures.

1.1 Background

The USA government has a set of laws and regulations designed to promote effective use of information technology and to ensure that all qualified vendors of information technology equipment and services have a fair chance to compete for government business. One law, commonly called the "Paperwork Reduction Act" (PRA), carries the broad direction that, "Each agency shall be responsible for carrying out its information management activities in an efficient, effective, and economical manner, . . ." (PRA Section 3506). The Office of Management and Budget (OMB), an executive branch agency with broad government wide responsibilities, issued OMB Bulletin A-130 (A-130) giving all agencies guidance on how to implement the PRA within their individual governmental mandates. Each agency, in turn, is expected to develop and enforce sound information resources management policies and practices based on A-130.

The authors arrived to direct an organization responsible for agency wide information systems QA. A reasonably complete set of documented policies and practices was already in place. However, the program was generally reactive - reviewing and criticizing systems projects only after major problems had developed. We set out to change the paradigm by:

- Educating the QA staff, systems developers, and executives about the importance of working on improving inputs and processes so that outputs did not have to be corrected so often;
- Measuring the specific problems in the agency and working on correcting them; and
- Ensuring that top executives were aware of the progress of their information technology initiatives in terms that allowed them to make sound business decisions.

2. QUALITY ASSURANCE METHODOLOGY

Quality assurance is defined as the activity or set of activities that provides "the evidence needed to establish confidence, among all concerned, that the quality-related activities are being performed effectively." (Juran and Gryna)

The success of any quality assurance program requires that four key elements exist - infrastructure, personnel, methodology, and metrics.

If the quality assurance program is to perform its oversight role, then the quality assurance and project development organizations must be separate but equal, both functionally and organizationally. The quality assurance organization at our agency was in fact separate and organizationally distinct from the organizations developing systems. However, it takes more than lines on an organization chart for a quality assurance program to be considered equal to a development effort. Senior management must view quality as a proactive process, and the personnel responsible for ensuring quality efforts must be viewed as credible by their customer organizations.

Upon our arrival, our assessment of the existing organization revealed that the senior management responsible for our organization wanted a proactive QA program. However, the majority of the efforts by senior management and quality assurance personnel were reactive and after the fact, when little could be accomplished in terms of corrective action. We took steps to train the existing QA staff in project management and quality assurance. We hired additional staff with backgrounds in systems analysis and project management. Several members of the QA staff studied for and successfully attained the Certified Quality Analyst (CQA) certification. Further training on total quality management, measurements, and information engineering life cycle was conducted. As a result of quality management training, the QA staff came to view the owners of the systems as internal customers. This allowed them to present quality assurance activities as services from which they would benefit. Gradually the quality assurance staff was seen as helpful to the project management and development staffs.

Concerning the methodology and the metrics, it was, and still is, our belief that our task was basically a simple one from a process point of view. Principles of QA for information systems are well developed and not really difficult to apply. We only had to construct a model that fit our environment and then convince others to cooperate. The QA methodology we employed borrowed heavily from other agencies' work, from our own experience, and from contributions of the QA staff. The QA methodology proved to work with both the traditional "waterfall" systems development techniques and the information engineering approach that we introduced into the agency. We also determined that QA works best if it is an integral part of the system development life cycle methodology.

Only major projects (those with an estimated system life cycle cost over $1 Million) were reviewed formally. This arbitrary distinction was made in consideration of limited personnel resources and because the amount of effort spent on oversight activities should be commensurate with the value of the project. Reviews were normally conducted annually. The reviews consisted of the following steps:

1. The owner of the system (a senior executive) was notified that the system was scheduled to be reviewed before the Executive Steering Committee (ESC) for Information Resources Management (IRM). The system owner was requested to submit written answers to a standard set of approximately seventy questions covering mission, needs, functional requirements, project management, quality assurance, costs, architecture, acquisition strategy, resources, training, interfaces, security, design, test and evaluation, transition, integration, and implementation.
2. The QA staff reviewed the answers and met informally with the system owner's staff to gain clarification.
3. The QA staff did a preliminary risk analysis and made recommendations based on the answers and other sources of information, and discussed them with the system owner's staff.

4. The system owner and staff met with the QA organization to formally review the risk analysis and recommendations. At this point there were occasionally disagreements, but they were the exceptions rather than the rule.
5. At the next quarterly meeting of the ESC the system owner made a presentation, followed by the QA Director's report and risk analysis. Recommended actions were discussed, and regular written status reports were subsequently submitted to the ESC.

2.1 Executive Steering Committee for Information Resources Management

This committee was composed of senior executives, most of whom headed major divisions of the agency and reported directly to the agency head. Their role was similar to a board of directors, who ensured that the IRM program was operating effectively and supported the mission of the agency. Decisions were reached by consensus, and major disagreements were generally settled by contending parties away from the actual quarterly meeting. Each major division had its own operating budget, and was thus free from any direct budgetary pressures from the other members.

The ESC had been in existence for some time before the QA methodology was implemented. However, discussions of individual system development efforts tended to be generalized status reports, and few specific action items were undertaken as a result of the meetings.

The central part of the QA Director's report to the ESC on each system was a simple matrix intended to give an instant impression of the project's overall level of risk, even if the executives were not interested in details (Table 1).

Table 1. Level of Risk Matrix

Information Categories	Level of Risk High	 Moderate	 Low
Mission			✓
Needs			✓
Functional Requirements		✓	
Project Management		✓	
Quality Assurance	✓		
Costs	✓		
Architecture		✓	
Acquisition Strategy			✓
Resources			✓
Training		✓	
Interfaces			✓
Security	✓		
Design			✓
Test and Evaluation			✓
Transition			✓
Integration			✓
Implementation		✓	

2.2 Consistency Of Reviews

The key to gaining the confidence of the ESC members was ensuring consistency of risk analyses from system to system and from meeting to meeting. Each senior executive had to be confident that his or her organization was being measured the same way and by the same criteria as were other organizations. The QA analysts developed a matrix to help make their evaluations as consistent and objective as possible (Table 2). The quality of the reviews was further reinforced by a critical meeting between the QA analysts and the QA Director prior to each ESC review.

2.3 Results Achieved

The risk matrix was used as a simple metric that could identify areas of risk that were common to several projects, as well as serve as an ongoing aid for the project to execute improvements in the moderate and high risk areas.

A sample of results follows:

- Projects redirected to simpler and less costly solutions;
- Security and contingency plans developed and tested;
- Configuration management controls implemented;
- Plans and budgets closely tracked - actual costs of project shown to management; and
- Structured information engineering approach followed with good results.

3. FINDINGS AND ACTIONS

A major benefit of consistent measurement of risks on a number of systems is that we identified problems that existed in many systems in many parts of the agency. By simply counting up the number of "high risk" evaluations over the course of a year we were able to see that project management and cost were consistent problem areas. Further analysis uncovered the reasons for this. We were then able to be proactive. We retained a consultant to teach courses in project management, which were offered to all managers involved in systems development. Concurrently, we issued a directive requiring that the status of major projects be reported monthly, preferably in electronic format from an automated project management tool. We also offered other courses in quality assurance, information engineering, and security. Concerning costs, we modified the acquisition orders to tie costs to a particular life cycle phase. While authorization could be granted for the entire project, the project could only progress and secure funds a phase at a time.

4. THE END

With the advent of a new government in 1993, management in the agency changed. The new management chose to discontinue the ESC. Therefore, the QA activities which revolved around the ESC and depended on support from senior executives, became dormant. Further complicating the QA efforts was the new administration's opinion that there were too many rules and regulations, whose very existence inhibited the efficient conduct of government. There is validity in that opinion. However, QA is not yet a part of the agency's culture, and is even considered part of the bureaucratic problem by "reformers" who don't understand the importance of process and measurement. In addition, there is the natural tendency of many entrenched bureaucrats to avoid any kind of accountability for their activities. That sort of bureaucrat is only too happy to shed any type of measurement mechanism. Therefore, nurturing a QA discipline suddenly became an almost impossible task.

5. CONCLUSIONS

We ourselves have experienced the truth that the quest for quality in information systems development is a journey with no end. We believed that when we started, but thought of the journey in terms of continuous improvements. Now it is clear that, at least in government, human and political factors may pose overwhelming challenges to the quality culture. Quality is a business issue, but political officials often have agendas that are not business oriented. In the USA, politically appointed executives tend to hold office for a short time. Thus, they typically are more interested in short term results than in long term business process improvements. Career government managers are also affected by the short term orientation of their political bosses, since they loyally strive to carry out the programs of the administration in power.

Because quality is common sense, management sometimes assumes that everyone wants to do and is doing the right thing. It follows then that quality assurance efforts are viewed as either redundant or superfluous. Unfortunately, this thought process makes dramatic assumptions about the workers' knowledge, skills, and abilities to perform tasks, and assumes that an omniscient individual is managing the entire process, connecting perfectly the myriad of tasks required to develop and deploy an information system.

Unfortunately, even though we sometimes refuse to believe it, senior management leadership and on-going support are the single major critical success factors for any quality effort. While some minor changes can be made and certain improvements affected, real change requires time and resource commitments, which senior managers ultimately control.

Middle managers have the largest adjustment to make in quality efforts. To be effective, they need to shed their past role of directing, managing, and controlling and replace it with one of leading, mentoring, facilitating, and empowering. The transition for these managers is difficult and the obstacles they can create are many. The authors found this to be true at their agency as well. Hence, senior management support was critical to keeping the middle managers focused on quality.

6. REFERENCES

A-130, (1993), "Revision to Office Of Management And Budget Circular A-130", Government Printing Office, Washington

Juran, J.M., and Gryna, F.M., (1993), "Quality Planning and Analysis", McGraw-Hill, New York

PRA, (1988), "Paperwork Reduction Reauthorization Act of 1985, Title 44 USC", Government Printing Office, Washington

Table 2. Level of Risk Assignment Matrix

INFORMATION CATEGORY	LEVEL OF RISK		
	HIGH	MODERATE	LOW
Mission	• Not identified in the 5-Year Plan • Not clearly defined	• Clearly defined, but needs to be better documented	• Detailed mission statement relating to 5-Year Plan • Clearly defined and well documented
Needs	• Not clearly defined • Not well documented	• Clearly defined, but not well documented for the functional areas	• Clearly defined and well documented for each functional area
Functional Requirements	• Not clearly defined	• Defined, but not documented consistently or in one place, but project team members know the requirements	• Well documented, cohesive requirements that designers and developers understand
Project Management	• No project plan • Schedule not updated • No methodology	• Plan in place, but not current (except for resources and schedule)	• Plan in place, sound methodology, all elements up to date
Quality Assurance	• No QA Plan • QA Reviews not scheduled • No independent evaluator or QA Liaison assigned • No clear understanding of QA program	• Plan in place but lacks description of responsibilities and execution • Reviews scheduled not sufficient • Some understanding of QA Program • QA Liaison involved but not assigned	• Responsibilities and execution clearly defined • Detailed reviews scheduled • Clear understanding of QA program • QA Liaison involved in activities
Costs	• Costs unknown • No Cost/Benefit study performed	• Cost/Benefit study performed but lacks sufficient documentation as to how numbers were derived • Cost figures are estimates, with little supporting documentation	• Detailed Cost/Benefit study performed with documentation as to how numbers were derived • Cost figures clearly defined with supporting documentation

Table 2. (Continued)

INFORMATION CATEGORY	LEVEL OF RISK		
	HIGH	MODERATE	LOW
Architecture	• Technical architecture not defined • Alternatives not considered	• Technical architecture defined but incomplete for entire deployment • Alternatives available but not thoroughly evaluated	• Technical architecture defined for total deployment of system • Selected architecture derived from evaluation process
Acquisition Strategy	• No acquisition plan • Insufficient amount of time allowed for acquisitions • Delegation of procurement authority not addressed	• Options not considered • Contingencies for acquisition not provided	• Acquisition plans, options, and related delegations of procurement authority are in place
Resources	• Not thought out • Non-specific	• Identified, but not applied across project in even manner (resource leveling)	• Detailed resource mapping to project tasks in place • All resources identified and allocated across the project
Training	• No training plan in place • Training requirements not identified	• Training plan exists but is insufficient to ensure users are properly trained • Training identified but not scheduled	• Detailed training plan has been reviewed and approved • Training clearly identified and scheduled
Interfaces	• Interfaces to other systems not clearly defined • Interfaces not clearly documented	• Moderate documentation exists • Handling of data and interfacing procedures not addressed	• Documented permission to interface with system exists • Responsibilities well defined • Interfaces defined early in the requirements phase
Security	• Sensitive system and does not conform to guidelines • No security plan • No contingency plan	• Security plan lacks specifics • Contingency plan lacks specifics	• Security plan in place and approved • Contingency plan approved and/or in place (tested)

Table 2. (Continued)

INFORMATION CATEGORY	LEVEL OF RISK		
	HIGH	MODERATE	LOW
Design	* System not GOSIP/POSIX compliant * System not designed in accordance with standards * Cannot trace design back to requirements * No clear design proposed	* Design can be traced back to requirements in limited fashion * Feasible design proposed, but others existed and were not evaluated	* Design clearly mapped to requirements * Design proposal selected from design alternatives
Test and Evaluation	* Users/testers of system not identified * No test plan or scripts	* Test plan & scripts in place but lack sufficient detail * Users/testers tentatively identified * Performance standards not established	* Test plan & scripts are detailed * Users/testers identified and roles understood * Performance standards and benchmarks established
Transition	* No transition plan	* Transition plan exists but does not clearly define procedures * Transition plan not evaluated by QA team	* A detailed transition plan has been reviewed and approved by QA
Integration	* Integration issues have not been addressed * Responsibility for integration issues has not been defined * Not integration plan prepared	* Integration issues are not addressed in sufficient detail * Integration plan not evaluated by QA Team	* A government individual has been assigned and issues have been addressed in sufficient detail * Integration plan prepared and evaluated by QA Team
Implementation	* No implementation plan	* An implementation plan exists but lacks sufficient detail * Implementation plan not evaluated	* A detailed implementation plan has been reviewed and approved by QA

MEASUREMENT OF SEX DEPENDENT QUALITY CHARACTERISTICS

Vlado Venuti[1], Peter Kokol[2], Ivan Rozman[2]

[1]TP KVIK Maribor,Gosposka 8-10, 62000 Maribor, Slovenia
[2]TF Maribor, Smetanova 17, 62000 Maribor, Slovenia

Abstract

The most important part of the majority of recent computer software systems, according to the users view, is the user interface. In this paper the authors present two new concepts of user interfaces: the sex dependent user interface design and an interface quality assessment metric for controlling that design. In searching for an appropriate metric to support above task we didn't have much success, so we decided to construct a new one. For that purpose we selected the Checklands soft system methodology and that's the third contribution of our paper.

1.INTRODUCTION

Measurement technology serves as a foundation for all scientific and engineering disciplines. The benefits of using measurement are well recognised, as are the costs. Almost all progress in physics, biology, chemistry and other traditional scientific disciplines, has occurred through interaction between measurement of objects and events in the real world and their abstractions in the world of models and explanations. On the other hand, the application of measurement technology to software engineering and science is relatively new, and therefore its benefits are not recognised by most people who work with computers. The development and use of adequate measurement of software and its development process are essential, however, to the production of cost-effective and reliable software.

Recent research (Rozman 91) has shown that about 80% of world software production is devoted to business related software systems (BSS). In the past most of these systems were written in COBOL, but today we can observe a great attitude shift towards end user computing and therefore to employment of user friendly application oriented computer languages (4th generation languages). The second characteristic of BSSs is that the user interface is, at least from the end user's point of view, theirs most important part. But according to new findings in "mind and brain research" there are some cognitive variations (Kimura 92) between the sexes. It seems that (Kimura 92) women outperform men at rapidly identifying matching items, verbal fluency, arithmetic calculations and in recalling landmarks from a route. Men, on average, have an advantage in tests that require the subject to imagine rotating an object or manipulating it, mathematical reasoning, in navigating their way through the route and in target directed motor skills. Similar findings were also discovered in the computer field by other researchers. For example Canada (Canada 91) reports on different technology - related attitudes, behaviours and skills of women and men; Abouserie (Abouserie 1992) reports significant differences between women and men in achievement in computer assisted learning; Lage (Age 1991) finds out differences in interests between sexes; and similar results were obtained by many other authors (i.e. Arch 1989; Cambell 1989, Smith 1987). Above variations suggest that man and women may have different interests and capabilities independent of societal influences. Thereafter it seems quite reasonable that user interfaces intended for women should be different from the interfaces intended for men. Such an interface is called a sex dependent user interface (SDUI) in our paper.

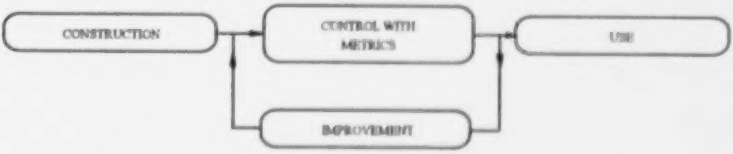


Figure 1.Use of metrics for development of software applications

As mentioned above it is a guarantee for progress in user interface design in general and especially in SDUI design to estimate its quality using an appropriate metric (see Fig. 1.). But there is a lack of published research in this field. It is our belief that conventional metrics which did not fulfil expectations at all, will completely fail in dealing with the above problem, so new metrics have to be developed.

1.1.Some problems

There are very few published reports on efforts and schedule figures for incremental development anywhere, let alone for implementations using a fourth-generation language. Boehm (Boehm 81) gave six examples of incremental development effort and schedule, but only one relates to a business application. It is neither possible to find any useful neither reliable figures for 4th generation development productivity. The one exception is the article written by Verner J.and Tate G.(Verner 88) who have developed an estimation approach based on Cocomo and function point analysis for the 4th GL called ALL. We didn't succeed to find any reference assessing the quality of the user interface according to sex characteristics in computer literature, but we found some useful hints in more general literature mentioned above.

The second problem with conventional metrics is the fact that they provide different measures for the same program. For example, a long program with few decisions and loops has a large lines-of- code (LOC) count or Halsted's E measure, but a small McCabe's cyclomatic complexity V(G). Therefore it is reasonable to combine them into a single metric. This combination should smooth differences and give more accurate results. As a possible solution we propose a hybrid metric. (Kokol 89).

1.2.Aim and Scope of the Paper

In this paper authors present two new concepts of user interfaces:

- the sex dependent user interface design and
- an interface quality metric for controlling that design.

In searching for an appropriate metric we did not have much success, so we decided to construct a new one. For that purpose we selected the Checklands Soft System Methodology (Checkland 1981, 1990) and that is the third contribution of our paper. In following sections we first give an overview of Hybrid metrics approach, and next we describe the Checklands Soft System Methodology. In the subsequent sections we present the use of SSM in the construction of SENSE metric.

2.HYBRID METRICS APPROACH

It can be argued (Kokol 89) that in situations where we would like to measure some characteristics of software systems there is a set of objects O on which a set of measurements with metrics Mi can be performed. For every object Oj, therefore, a subset of metrics values Mij (i = 1...n) is obtained, where n is the number of metrics. To combine them into a single metric and avoid difficulty with different units (for example, Halsted's E is expressed in [mental discriminations] and McCabe's V(G) in [linearly independent paths]) the following model is proposed:

$$H\_M_j = \frac{(M_j + R_{M \Leftrightarrow M1} M_{M1j} + R_{M \Leftrightarrow M2} M_{M2j} + \ldots + R_{M \Leftrightarrow Mn} M_{Mnj})}{1 + R_{M \Leftrightarrow M1} + R_{M \Leftrightarrow M2} + \ldots + R_{M \Leftrightarrow Mn}} \qquad (1)$$

where

$$M_{Mij} = f_i(M_{ij})\,; \quad i = (1 \ldots n)\,; \quad j = (1 \ldots k)$$

M is the basic metric,
Mi are other metrics of interest,
fi are functions to calculate basic metric values Mj from other metrics Mi,
n is the number of metrics of interest (except the basic metric),
and **k** is the number of objects O.

Functions fi and coefficients $R_{M \Leftrightarrow Mi}$ are calculated via regression analysis between the subset of metric values Mij and the subset of basic metric values M1j for each pair (M,Mi) separately. The experimentation with various regression models has shown that the best results are attained with a linear model, which is also easiest to compute. Earlier equations present a weighted average of selected metrics. If there is no correlation between the basic and other metrics then the second metrics has no impact on the final score. On the other hand, if there is a perfect correlation between the basic and other metrics then this equation becomes an equation for computing the ordinary average value. Actual results lie somewhere between these two extremes.

CHECKLANDS SOFT SYSTEM METHODOLOGY (SSM)

Following recent research which shows that the conventional scientific approach is not successful in solving unstructured and complex real - world problems (Checkland 1981, 1990; Flood 1992, Trapl 1994) we have selected the SSM as an appropriate construction methodology. The SSM uses system ideas to formulate basic mental acts of four kinds: perceiving, predicting, comparing and deciding on actions.

SSM begins with a soft and unstructured expression of a problem situation, and continues with the definition of some human activity systems which seems relevant to the problem situation, named root definitions. To be efficient every root definition should include six crucial characteristics, namely Customer, Actor, Transformation,Weltanschauung, Owner and Environment (CATWOE elements). Root definitions are used next as the basis for making conceptual models of systems being selected. After the model is built it is compared with the situation perceived. The primary aim of the comparison is to comprise a debate, discussion or argument out of which suitable changes can be made. Once the debate stages have revealed possible changes, then the new problem situation becomes that of implementing these changes in the real world. The SSM never solves problems out of existence, it is just the never ending learning process which should possibly improve the problem situation, and enables with the gained new knowledge to start another cycle of learning. The authors have successfully used the SSM in various applications related to software and information system design and control (Kokol 91, Kokol 92, Kokol 93, Kokol 94).

3.SENSE METRIC

As mentioned in the introduction the SENSE metric was developed with the help of the SSM. The study presented in following paragraphs was performed by the methodology presented by Checkland in 1981 (Checkland 1981). Currently we are trying to improve the obtained results by using the SSM in more modern form (Checkland 1990, Kartowisastro 1994) including the 5E analysis and our achievements will be presented in forthcoming papers.

3.1.Root Definition

Taking the following systemic definition *to evaluate a system means to measure its parameters in terms of well known metrics using standard instruments so as to analyse it against standard accepted benchmarks*

as the basis the next root definition of SENSE metric was constructed:

SENSE should enable the user to analyse selected user interface (UI) characteristics in the manner to transform them into a single number which after a comparison with proposed standard border values (see Fig. 2.) should clearly indicate for which sex the UI is more appropriate and if it is appropriate at all by formal (automatic) examination of UI programming support and the analysis of users feelings about the UI taking into account "state of the art " in IT, ethics constraints, law etc. in the manner to achieve user satisfaction of both sexes..

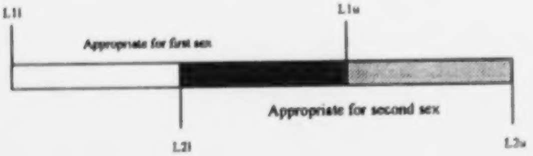


Figure 2.: Standard border values - SBV (L1l, L1u, L2l, L2u)

CATWOE elements of above root definition are shown in Table 1.

Table 1. The CATWOE elements of the SENSE root definition

CATWOE	DESCRIPTION
Customer	Users of the BSS
Actor	BSS and SENSE designers
Transformation process	UI characteristics --> single value representing that characteristics
Weltanshauung	Using SENSE in UI design is very benefiting in the
manner	to achieve user satisfaction of both sexes
Owner	BSS and SENSE designers
Environment constraints	The situation in BSS design (see the Introduction),
ethics	constraints, law, "state of the art in IT"

As we can see from the table 1 there are two owners of the system, particularly BSS UI designers and SENSE designers. In such a case it is usual to construct two root definitions and two conceptual models, but fortunately it was possible to incorporate requirements and weltanschauungs of both groups into a single root definition.

3.2. The Conceptual Model

The conceptual model based on above definition (without the control and 5E subsystems) is shown on Figure 3. The submodel for the BSS UI designers (or in other words the users of SENSE metric) is presented inside the doted rectangle. As we can see it is simple - it consist out of only two main activities: measurement (analysis) and comparison. The input to the submodel is a BSS UI and the output is the class to which it belongs (there are four main classes: inappropriate, appropriate for male users, appropriate for female users or appropriate for both users).

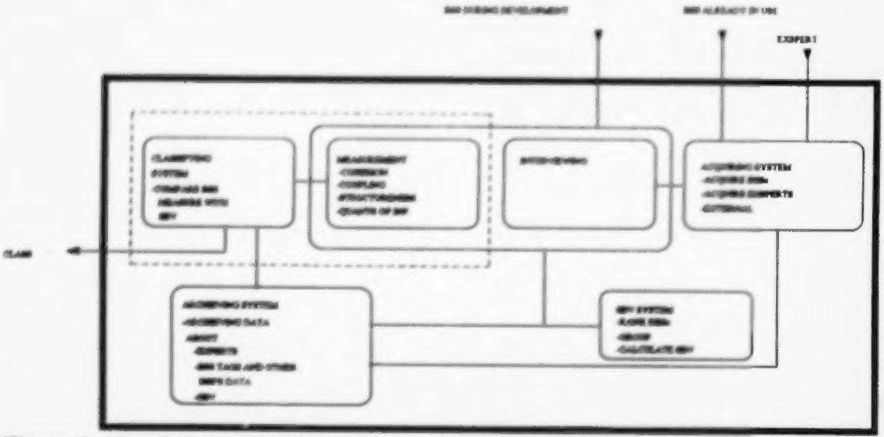


Figure 3. The Conceptual Model (SBV stands for standard border value - see Figure 2)

The main model consist out of five subsystems. The tasks of archiving, classifying and acquiring subsystem are indicated in the figure 3 and the role of taging and standard border values (SBV) system is presented in more details in next paragraph. The taging system incorporates two activities: the first - the measurement enables one to measure the UI characteristics and represent theirs "quality" as a single value (BSS measurement); the second - the interviewing is used to empirically evaluate the BSS UI by interviewing the BSS users and also independent experts. In the standard border values subsystems empirical marks are used to rank BSSs separately for male and female users using the Andersons method (Anderson 89). Each group of BSSs is then divided into three percentile subgroups. The first subgroup represents the BSSs from 1st to 5th percentile, the second from 6th to 95th percentile and the third the rest. The average of BSS measures the first subgroup represent the lower border (Lil), the average of the third subgroup the upper border (Liu) and the average of the second subgroup the optimum of appropriateness of the BSS UI. These border values are calculated for male and female group separately and represent the basis for classifying the BSS UIs into four classes enumerated previously. In that manner the SENSE can be used for three different purposes:

1. **find equalities:** to design an UI appropriate for both sexes
2. **find differences:** to design either an UI for women or an UI for man
3. **enable free choice:** to design three different Uis - one for women, one for men and one for both, and to leave the user to choose her/his favourite one.

3.3.The Measurement for 4th generation Languages

The measurement activity measures the efficiency and effectiveness of UI in terms of quantity-of-information- per-screen, differentiation between screens (cohesion and coupling) and structuring. We had no problems at all with the measurement of duration time, error rate and response times. Information-per-screen was measured as information-quants. One quant of information is defined as the quantity of data input between two confirmations (Enter or Return keys). Difference between screens was measured similar to measurement of coupling an cohesion of programme modules (Conte 1986). Cohesion was measured in terms of relationship between data on one screen. If all data on the screen belongs to the same entity, the cohesion is high. If data on one screen are attributes of two or more related entities, this is medium cohesion and if data do not relate directly, this is called low cohesion. This can be presented by the formulae:

$$OH_i = \frac{(W_e N_e + W_{re} N_{re} + W_u N_u)}{(N_e + N_{re} + N_u)} \qquad (2)$$

where:

COH$_i$ = cohesion for one entry screen
N$_e$ = number of data belonging to the same entity
N$_{re}$= number of data (together) which belong to related entities
N$_u$ = data belongs to unrelated entities
W = weights (currently arbitrary set by the researchers)

And finally:

$$COH = \Sigma(COH)_i \qquad (3)$$

Coupling between two screens is measured by counting how many data items appear on two different screens and belong to the same entity (Ns) or related (Nr) entities. This can be presented with following equation:

$$COU_i = W_s N_{si} + W_r N_{ri}; \qquad i = (1, 2, \ldots, n\text{-}1) \qquad (4)$$

where

n = number of screens

The cumulative coupling for one screen is calculated as the arithmetic average of partial couplings.The measuring of structuring (S) is based on the graph theory. Here we can assume that every single input screen is presented as a graph node. In our case best is, if the number of paths between various nodes is as high as possible, but the length of each path must be as short as possible.

$$S = \frac{No.\ of\ \ paths}{No.\ of\ \ screens} \tag{5}$$

At the end all above metrics have to be combined using the hybrid metric.

3.4.First Results

Recently we succeed to analyse a couple of BSS and the results we got are shown in table 2.

Table 2. Current Standard Border Values

	Male	*Female*
[illegible]	02.13	9.47
[illegible]	11.89	15.81
Optimum	07.65	12.54

4.CONCLUSION

With the use of this new developed SENSE metric we have controlled the development of the software in our company KVIK. This software is developed mainly to control the financial flow of data through all stages of process from incoming of goods to selling them. Most important in this process is to input proper data on very beginning, when goods came in. This job is done mostly by women. But anyway we had to design two different kinds of interfaces for data input. That was possible only with use of our new developed metric. With such approach we have enormously improved the data input process. The consequences of this improvement are mainly that the error rate at the input point has dropped from early 11.4% down to 3.9% and the time needed for input is shorter nearly for one quarter. But that are not all of the improvements we reached with use of our method. The whole software development process is done more efficient, effective, with less resources needed and with lower error rate.

In the SENSE design process the SSM was used as the main intellectual tool. It is natural to ask question such as: *Does it work?, Is it good? Is it more usable then approach X?, etc.* But, as Checkland (Checkland 1981) pointed out such questions can not be answered. There is in principle no way in which it could be proved or disproved that using the SSM was "*the best*" way to design SENSE. However, the success of our work in KVIK and the results of some recent projects show that the use of SSM was successful. SSM meat our requirements, in the sense that it was relatively easy to learn, apply and adapt for our needs and culture. It was used more in the Mode 2 manner (Checkland 1990) and enabled us to carry out the SENSE design process in systematic, systemic and human intensive way.

Nevertheless, it must be said, that the since now the performed process represents just first few phases toward the final form of SENSE. Currently we are improving our metric and its use process by "upgrading" to the SSM as introduced in Checkland (Checkland 1990 and Kartowisastro (Kartowisastro 1994) and by using the Metaparadigm (Kokol 1994). Our main goals are to enable one to asses more UI quality factors, to automate the measurement process defined with the conceptual model, to improve the root definitions and conceptual models and to prove them according to 5Es analysis.

5.REFERENCE

Abouserie R., Moss D., Barasi S. (1992), Cognitive Style, Gender, Attitude Toward Computer Assisted Learning and Academic Achievment, *Educational Studies*, Vol: 18(2).

Anderson E. E. (1989), A Heuristic for Software Evaluation And Selection, *Software Practice and Experience*, 19-8.

Arch E., Cummins D. (1989), Structured and Unstructured Exposure to Computers: Sex Differebces in Attitude and Use Among College Students, *Sex Roles*, Vol: 20 (5-6).

Bailey J. E. (1987), Computer Based System to Measure and Analyze Computer Users Attitudes, *Policy and Information*, 11-2.

Boehm B.W. (1981), Software Engeneering Economics, Prentiece-Hall, Englewood Cliffs, N.J.

Campbell N. (1989), Computer Anxiety of Rural Middle and Secondary School Students, *Journal of Educational Computing Research*, Vol: 5(2).

Canada K., Brusca F. (1991), The Technological Genfer Gap: Evidence and Recommendations for Educators and computer-based Instruction Designers, *Educational-Technology-Research and Development*, Vol: 39(2).

Checkland P. (1981), Systems Thinking, System Practice, John Wiley, Chichester.

Checkland P. Scholes J. (1990), Soft System Methodology in Action, Chicester, Wiley.

Conte, Dunsmore, Shen (1986), Software Engineering, Metrics and Models,The Benjamin / Chummings, Menlo Park, CA.

Denning P.J. (1993), What is Software Quality ?,EDITORIAL department of the Communications of the ACM", vol.35, No.1.

Flood R. L., Jackson M. C. (1991), Creative problem Solving: Total System Intervention, Wiley.

Kartowisatsro H., Kyiochi K. (1994), An Enriched Soft System Methodology (SSM) and Its Application to cultural Conflict Under a Paternalistic Value system, *System Practice*, Vol: 7(3).

Kimura D. (1992), Sex Difference in the Brain, *Scientific American* Vol: 267-3.

Kokol P. (1989), Application of Spreadsheet Software in Software Engineering Measurement Technology, *Information and Software Technology*, Vol: 31, pp. 477 - 485.

Kokol P., Stiglic B., Zumer V.(1992), Software Design Tool Evaluation in Context of a metaparadigm, Proceedings of AQSDT'92, IEEE CS Press.

Kokol P., Stiglic B., Zumer V.(1991a), New Evaluation Framework for Assesing the Reliability of Engineering Software Systems Proceedings of RRES'91, Elsevier.

Kokol P. (1991b), A New Microcomputer Software System Evaluation Paradigm: The Medical perspective, *Journal of Medical Systems*, 15-4.

Kokol P. (1994), Software System Design With The Metaparadigm, Cybernetics and System 1994, World Scientific, Singapore.

Lage E. (1991),Boys, Girls and Microcomputing, *European Journal of Psychology of Education,* Vol: 6(1).

Rozman I. (1991), Quality in Software Engineering, QPP'91 Proceedings, Maribor.

Smith S. (1987), Computer Attidues of Teachers and students in relationship to gender and Grade Level, *Journal of Educational Computing research,* Vol: 3(4).

Trapl R. (1994), Cybernrtics and Systems Proceedings 1994, World Sciemtific, Singapore.

Verner J. and Tate G. (1988), Estimating Size and Effort in Fourth-Generation Development, *IEEE Software*, Vol: 5,pp.15-22

Waern Y (1989)., Cognitive Aspects of Computer Supported Tasks, John Wiley & Sons, Int.

SOFTWARE QUALITY MANAGEMENT AS A BASIS FOR COURSE DESIGN

Alastair Monger, Vlada Petruv, Margaret Ross, Geoff Staples, Ian Tromans

Southampton Institute, Information Systems Division, East Park Terrace, Southampton, England

Abstract

The paper describes the strategies and content of the computing courses, both full-time and part-time at Southampton Institute, and how they respond to the needs of local industry. The theme of quality is central to these courses, and the manner of its integration is discussed. The wider distribution of computing ideas, beyond these courses to the local companies, are also considered.

1. INTRODUCTION

In May 1993, the UK Government produced a White paper, 'Realising Our Potential' (HMSO, 1993)) that encouraged the spread and understanding of all aspects of technology, including computing, within the academic sector and into the commercial and industrial sectors, in order to promote the quality of life and economic advantage. At Southampton Institute, there are a series of related full-time and part-time courses, targeted at the computer professional and the end users. These include BSc degrees, suitable for potential computer professionals and end users (three years full-time or four years full-time, with the third year spent working in industry, or four or five years part-time), Higher National Diploma courses, of a more vocational nature (two years full-time or three years part-time) MSc degree (one year full-time or two part-time) and professional courses leading to the British Computer Society Part I examinations (three years via evenings only, with a one year fast path for day release and remote students), and to the BCS Part II examinations, leading to corporate membership (one or two years via evening classes or remote learning). The part-time students attend on one full day each week, or one afternoon and two evenings per week, or two evenings per week. The Remote Learning students receive a weekly package of notes, exercises and examples, via the post, and they return their work in the same manner, but they can also communicate with their lecturers by telephone or by fax. There are remote students on the BCS courses in all parts of the U.K., and in many parts of the world, particularly the

Far East. The majority of the full-time students are aged between 19 and 25. The part-time students are approximately five years older than their full-time counterparts. Nearly ten percent of the mature students are over the age of forty. The fees of the full-time students are normally paid by government grants topped up by loans, whereas the fees of part-time students are normally paid by their employers or directly by the students themselves.

2. INDUSTRIAL LINKS

The design of all new computing courses involves computing industrial advisors, together with market surveys directed at local potential employers and of past computing students, to test the potential applicability of the course. The validation events, to ensure that the course has been correctly prepared, would always include external members from the computing industry, who would cross examine the course team, who prepared the course documentation. For established computing courses, the links with local firms are strengthened by the participation of part-time students, and the placement of third year students in a company, for the sandwich year. Final year full-time and part-time students' projects normally address real problems, involving local companies. The requirements of the local employers, both in the short and long term (Ross and Staples, 1994), must always be a major consideration in the design of computing courses.

3. COURSE CONTENT

Software quality is an underlying theme in all our computing courses, as quality can be viewed as an umbrella, encompassing good practices of analysis, design, coding, testing, project management, and even security and risk management. Quality also included concepts such as user satisfaction, portability and inter-operability.

All the courses include various analysis and design methods, including SSADM (Structured Systems Analysis and Design Methodology) (NCC, 1986,1990; NCC, 1988), which is the chosen method of the UK government. The strength of the method is that it was developed, using the best practices of many of the previously established structured methods, so SSADM provides a natural base for comparison with other methodologies. Another feature is the emphasis placed on cross-checking between the various techniques used in SSADM, to identify potential problems at an early stage.

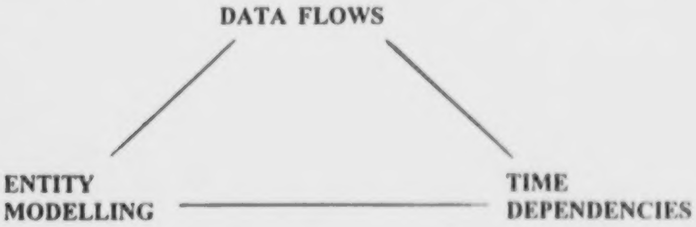


Figure 1: Views of SSADM

The interaction between the three views of the computer system, within SSADM, (Figure 1) based on the three concepts of data flow (data flow diagrams), relationships (entity modelling) and time (entity life histories) help to ensure confidence in the requirement analysis and the final design. The major strength of this method is the commitment to quality reviews and quality issues at all stages of the method, which provides natural links to the other aspects of quality. The observance by the students of standards and more important the understanding of the need for, and the full implications of such aspects of the appropriate forms and standards, will make the students more adaptable to the use and observance of the employers standards. This is essential for employers wishing to achieve or to retain quality certification, such as ISO 9001, or quality initiatives such as Total Quality Management (TQM). The observance of standards and procedures by employees also provides a firm framework for efficient and effective future maintenance and enhancement. The estimating features of SSADM allow the students to appreciate the costs, in terms of time and money, between different variations of the systems boundary, and at a later stage, between alternative designs. Some of the computing courses have had the Structured Analysis and Design modules accredited by the UK central authority, as recognised training for the Certificate in Proficiency in SSADM (Version 3). This allows students with the required one year relevant practical experience of analysis and design, on the completion of this module, in addition to their course examinations, to take the UK centrally set Certificate of Proficiency examinations. This consists of a three hour no choice written paper, followed by an oral examination. These accredited SSADM modules are particularly useful for part-time students and their employers. Students who passed this examination have normally received promotion or have been able to move to better jobs immediately or within a few months of being awarded their Certificate of Proficiency. The full-time and part-time students, without the necessary practical work experience, can still state on their Curriculum Vitae that they have attended an accredited SSADM course, and this could provide a major financial saving for their future employers. Southampton Institute also runs two week accredited intensive courses for industry, leading to this examination, which is a requirement now, for analysts working on certain projects.

4. PROJECT MANAGEMENT

The inclusion of the Prince project management methodology (CCTA, 1990; Bentley, 1992; Bentley, 1991), with its total commitment to quality, is also included in the majority of the computing courses. The methodology Prince (PRoject IN Controlled Environment) is the chosen project management of the U.K. government, and is used, either in part or in its entirety, both within and outside government projects.

The impetus to expose students to Prince from 1990, and previously its predecessor, Prompt, was partly due to the geographic location of Southampton surrounded by employers who are influenced by the need for quality accreditation, and also the need to change students attitudes to project management, by making them aware of the wider implications of project management, and the need to embed quality into the framework of a project from the beginning.

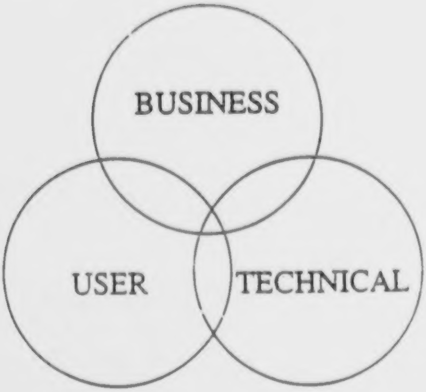


Figure 2: The Three Views of Prince

The interaction between the three views of a project (Figure 2) are considered. Representatives of the User (Client/Customer), the Business (Finance/Management) and the Technical (IS department) form the permanent membership of three on the Project Board, and at a lower managerial level, from the permanent membership of three on the Project Assurance Team. There is a permanent project manager, supported by a variety of stage managers, one for each of the different stages of the Prince project. The reporting structure and the variations of organisation provide a useful source of

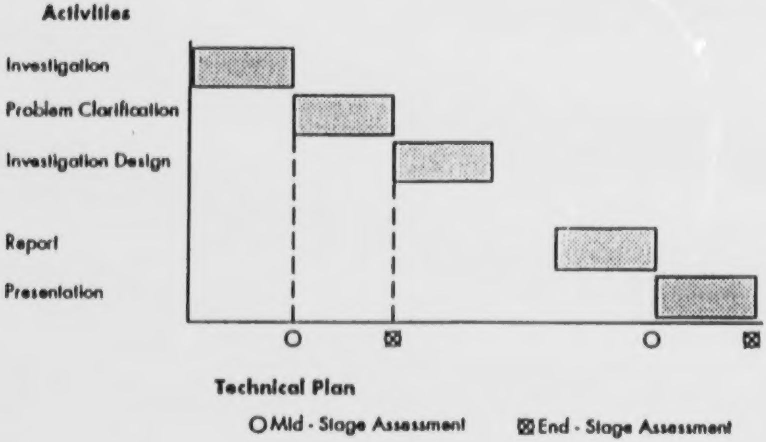


Figure 3: Technical Plan

discussion, particularly with part-time and mature full-time students, who are encouraged to describe their experiences and to identify where the Prince approach might be or might have been of benefit to their companies. As project management is not normally taught in the first year of a course, the use of the various project plans are often based on student projects completed in the previous year. The students are encouraged to work in syndicate groups of usually three, using their previous experience of the student project, to prepare full project plans for future students to complete that student project. This would involve identifying units of work, each with an identifiable end product, and the interdependences between them. The syndicate group would produce the technical plan (Figure 3) (similar to a Gantt chart) which shows the different stages, end stage assessments, quality assurance reviews, and the frequency of the management reports.

They would also produce the Resource Plans (similar to a spread sheet), indicating the costs, expected and actual, on a stage by stage basis. The aim is to encourage the syndicate to balance the loading of the work, within given constraints, concerning the relative experience and comparable costs associated with the proposed future students who could use the project plans. Requirements for an early completion time and low costs are also imposed.

The final project plans are presented to the other students, using role playing of a formal quality assurance review. This underpins and emphasises the benefits of the formal review process, and usually generates considerable enthusiasm among the syndicate groups. Pat students who have undertaken these exercises have found them extremely useful, when working on real-world projects, both in gaining confidence and being aware of the various conflicting factors and risks involved in managing a project. The introduction to Change Control is well defined within Prince, and the need for such procedures are related to the programming modules of the course. The roles of the various members of the Project Assurance Team, to clarify the users request then to evaluate the technical involvement, and then to provide a costing for the proposed change, prior to a decision being made by the Project Board, are discussed. The experience of the students is particularly useful, as a series of disaster stories normally arise, resulting from poor change control procedures. The students can use a simplified form of these procedures, to handle requests for change from the customers of their projects, or as a result of prototyping. These simplified documents can form part of the final documentation of the students' projects. The need for a Project Support Office, and its role within the Prince methodology, is normally discussed. This is seen as a home for some members of the Project Assurance Team and the configuration manager. Students, both with real world experience and with only experience of syndicate programming exercises, easily understand the need and benefits of an efficient configuration management procedures. This leads naturally into discussion of links with the testing and programming modules. The modification of Prince for small projects and for Non IS projects (CTAA, 1991) are discussed. The wider benefits of management visibility of a project, early warning of problems, identification of potential risks and user participation, within the Prince methods are discussed. The nature of this non-proprietary project management method underpins the concepts of many of the other modules with the courses, and emphasises the need for quality within software projects.

5. SOFTWARE QUALITY

The need to provide students with an understanding of the concepts of software quality, is a major consideration of all the courses. This results from the requirement of many companies, both large and very small (a three person firm) to achieve and maintain a quality accreditation, or to improve their profit by removing the need for expensive and delaying corrections of errors, or even to protect themselves from future legal litigation for failures of software. A survey of fifty local companies was undertaken (Ross and Staples, 1993), to identify the employer's views of the inclusion of quality related topics to computing courses. The employers were from five equally sized groups, consisting of finance (including insurance companies, pension firms, banks), defence (such as the military services, companies working on defence contracts), utilities (such as producers or distributors of different forms of energy, health care, communications), software houses (including producers of computer software or hardware) and other (including those not in the other categories).

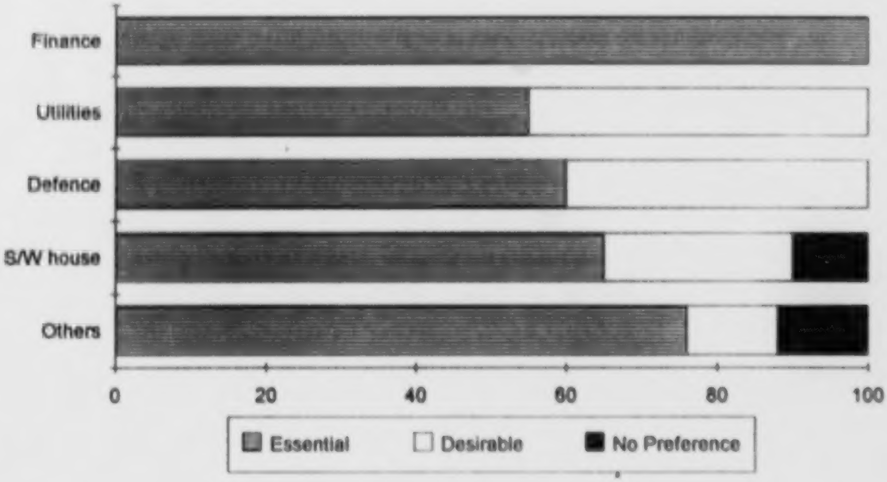


Figure 4: Survey of Employers on Quality

Almost all the employers felt that quality (Figure 4) and quality standards (Figure 5) should be included, as essential or highly desirable within computing courses. [illegible] inclusion of methods such as SSADM and Prince, with the emphasis on quality, was generally felt to be of benefit to the students, by the employers. The students are made aware of the externally audited accreditation schemes, such as ISO 9001, BS5750 or EN29000, and the similar scheme designed specifically for the computer industry, TickIT.

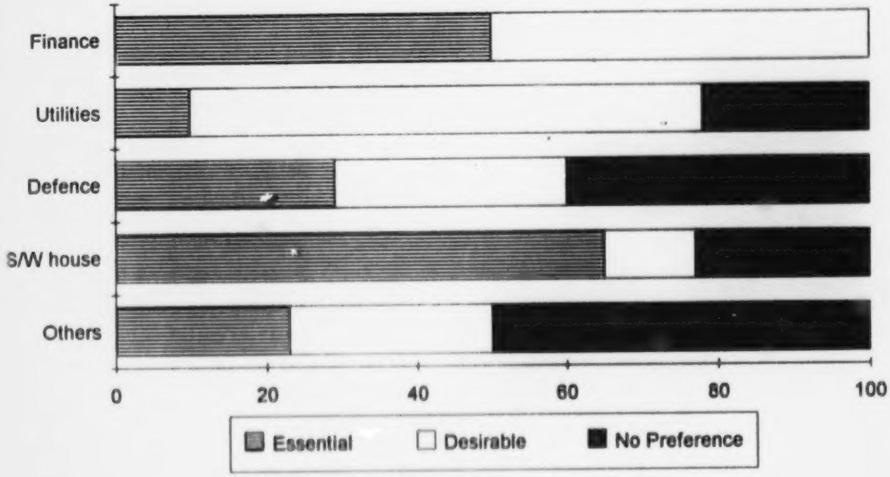


Figure 5: Survey of Employers on Quality Standards

Alternatively, self regulating approaches such as TQM (Total Quality Management) and the Software Engineering Institutes Capability Model, with five levels of achievement are also discussed. Many of the part-time students have experience of one or more of these approaches.

Students are made aware of the need for metrics, to underpin the various approaches to quality. The practical problems currently associated with the use of metrics, whether to measure quality, the productivity of employees or the complexity of programs are discussed. Part-time students are encouraged to consider which of the metrics, not currently being utilised by their employers, could be of benefit to their company.

There is a constant link created between the elements of the course, and current practices of local firms and of the employers of the part-time students. This provides a basis for lively discussions in addition to providing a means for examples of good practice to be exchanged between employees of different companies. By encouraging part-time students to base course assessment exercises on their own companies, it provides a framework for internal reports from these students to their managers. An example was a report on the use of metrics to reduce the complexibility of code, which simplified testing and assisted with maintenance. This has resulted in promotion for various part-time students to implement the proposed changes of procedures within their companies, so adding to the effectiveness and efficiency of these companies.

6. PROFESSIONAL LINKS

Part-time and full-time students are actively encouraged to participate in activities of the local branch of the British Computer Society. This provides the students with the opportunity to listen to outside speakers, who are authorities in their field, discussing issues relevant to the computer industry. Meetings are often held at Southampton Institute, or on the premises of local firms. Meetings, which have been held in 1994, include such topics as the identification and creation of the skills for today's and tomorrow's computer professional and user; the latest developments in object-oriented analysis and design; data compression for text and documents; the experiences of quality auditors and auditees to achieve TickIT; and the awareness of ways to prevent computers damaging the physical health of a computer user. The students can also join the visits to local companies, which are arranged by the British Computer Society. These activities, beyond the students prescribed course, help to widen their knowledge and bring them into contact with the debates between computing practitioners from a variety of companies. These meetings and visits are felt to be very useful for employees of companies, as a means of providing them with an update on current ideas across a variety of computer disciplines, at no cost, and to provide them with a window on the activities and good practices of other companies.

7. CONCLUSIONS

It is felt that by considering the quest for quality as a theme within the computing courses, the links exist naturally with other modules, such as programming, analysis and design, and the management of projects. This encourages the students to take a holistic approach to the computing industry, whether as computer users or professionals. This view is mirrored by the breadth of topics addressed within the professional BCS meetings, which are viewed as providing an opportunity for the professionals of today and of tomorrow to receive regular updates, briefings on matters of concern, and to promote the development and maintenance of high quality computing systems.

8. REFERENCES

Bentley C, Introducing Prince, NCC/Blackwell, 1992.

Bentley C, Practical Prince, NCC/Blackwell, 1991.

CCTA, Prince Applied to Non-IT Projects, HMSO, 1991.

CCTA, Prince Guides, NCC/Blackwell, 1990.

HMSO, White Paper, 'Realising Our Potential' May 1993.

NCC, SSADM Developers Handbook, NCC/Blackwell, 1988.

NCC, SSADM Manual (Version 3 & 4), NCC/Blackwell, 1986, 1990.

Ross M, Staples G, Industrial Influence, Proc of ISTIP, UK, 1994.

Ross M, Staples G, Promoting Quality Awareness, proceedings of Software Quality Management '93, Southampton, UK, May 1993

Requirements engineering/ system specification

HYPEROBJECTS:
AN OBJECT-ORIENTED, 4TH-GENERATION SYSTEM PROTOTYPE

Alois Stritzinger

Ch. Doppler Laboratory for Software Engineering, Institut fuer Wirtschaftsinformatik
Johannes Kepler University of Linz, A-4040 Linz, Austria

Tel.: ++43-732-2468-9437
E-mail: stritzinger@swe.uni-linz.ac.at

Abstract

Object-oriented programming is currently revolutionizing application programming technology. Beyond flexible implementation techniques, object-orientation offers a very convincing thought model which allows modelling real world entities and their relations in a natural way. For this reason database research is also trying to benefit from object-oriented advantages. But for end users and 4th-generation developers, objects are still merely a marketing buzzword for graphical user interfaces.

In the HyperObject project we are trying to make general purpose, object-oriented data models visible and accessible for information system designers and end users. A primary goal is to make the object model transparent to the user. In this paper the basic concepts of HyperObjects and their implementation are presented. Based on experiences with a first prototype, we discuss useful improvements and outline our future plans.

Research Report
Keywords: 4th-Generation Systems, Information Systems, Object-Oriented Modelling, Data Models, User Interfaces

1. Introduction

Information systems are among the most important software applications. They are often characterized by a quite large database which can be accessed and manipulated by transaction-oriented programs. To enable end users to develop their applications themselves, or at least ease the programmer's job, so-called 4th-generation systems were introduced. Usually such systems include or connect to a (relational) database, a data model editor, and some kind of interface builder.

Hierarchical and relational data models have proven inadequate for storing complex data structures of special sophisticated (information system) applications, e. g., hypermedia, computer aided design and manufacturing systems. Object models seem to be more flexible, because they support natural modelling of real-world entities [Rumbough91]. Hence, object-oriented data bases are currently a heavily investigated and published topic in the database community (see [Kim89]). Nevertheless, it seems that object-oriented databases still have some way to go to become a practical technology.

In recent years object-oriented programming has found wide acceptance in the scientific community and has shown signs of success in practice too. Nevertheless, few attempts have been made to incorporate object-oriented concepts into 4th-generation systems to make the advantages accessible to end users. The main goal of the development of HyperObjects is to incorporate the object-oriented thought model (see [Rechenberg90]) into a generic, general-purpose information system in the internal data model as well as in the mental application model.

2. Basic Concepts and Terms

The information carrier or the end-user objects of our system are called ***Entities***. Entities either consist of named components which hold relations to other entities or contain atomic values. The network of entities and the relationships among them form a pure, structural object-oriented data model–everything is an entity. If HyperObjects were a complete database, it would be classified as structurally object-oriented.
(see [Atkinson89])

Each entity is described by and copied from a system- or user-defined ***Pattern*** which describes the entity structure. There exist three different kinds of patterns:

Basic Patterns

Basic patterns define atomic data like strings, integers, booleans, dates and times. A rich set of operations for these data types is provided by these basic (system-defined) patterns.

Object Patterns

Object patterns are user-defined and describe entities with an arbitrary number of attributes. Each attribute has a name and a type (i.e., a pattern) assigned to it. Since attributes can only hold relations of type-conform entities, the system is strongly, dynamically typed. Furthermore, each attribute is characterized by a key flag and a collection flag. The key flag defines the most significant attributes, which need not form a true key; the key attributes are used for visualization instead. Since HyperObject is purely oject-oriented, every entitiy has a unique id, of course. The collection flag defines the valence of an attribute. Normal attributes contain a reference to a basic or an object entity, collection attributes refer a collection entity which contains several elements.

Collection Patterns

Collection patterns describe containers which elements are references to other entities and can be accessed by an integer index. The container size (number of elements) may shrink and grow dynamicly. Collection patterns are automaticly generated by the system for each newly defined object pattern. All elements of a collection have to be type-conform with the object pattern.

The Patternspace

The object patterns have to be defined and can be modified by the user in a special design mode. Object patterns may be derived from existing patterns, causing the inheritance of the attributes of the superpattern. The collection of all patterns and the relationships between attributes and patterns form the patternspace. It is possible to add or remove attributes from an existing pattern. In this case all corresponding entities are updated automatically.

The Hyperspace

The network of all entities and their relationship form the hyperspace. Working in edit mode permits creating and changing entities. The browse mode allows just navigating and inspecting in hyperspace.

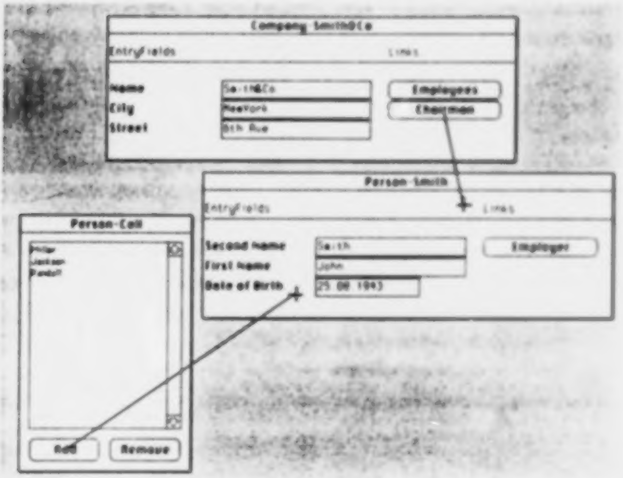
The Viewspace

A number of views serve for the visualization of single entities or small portions of the hyperspace. Views are either provided by the system (standard views) or can be designed by the user by means of an interface builder. The standard views show a single entity–either an object or a collection. Furthermore, two root views are provided. The user-root view shows a collection of user defined entry points into hyperspace. The system-root view displays a collection of all patterns, were each entry is linked to a collection containing all entities of a certain pattern (similar to relational tables).

3. Prototypical User Interface

Figure 1 shows three standard views: one collection view ('Person-Coll') and two object views ('Company-Smith & Co' and 'Person-Smith'). The standard object views contain entry fields for all basic pattern attributes and buttons for all object-pattern attributes. The key attributes are also displayed in the view's title. Pressing a button opens or brings to the front the related object or collection view. The standard collection views show a selection list containing the key attributes of their elements. Selecting a list element opens a view with the associated entity in a similar way as described above.

Pressing the Add button of a collection view creates a new entity, establishes the relationship, and opens the corresponding view. Pressing the Add button and dragging the mouse into an existing view establishes a relation to an existing entity. Similarly, pressing a button in an object view either opens a view, or–when dragging–establishes a relation to an open entity.


Company-Smith&Co
EntryFields
Links
Employees
Chairman
Person-Smith
EntryFields
Links
Employer
Person-Coll
Add
Remove

Name: Smith&Co
City: NewYork
Street: 8th Ave

Second name: Smith
First name: John
Date of Birth: 25.08.1943

Figure 1. Standard views

Figure 2 shows the two root views. The system-root view cannot be changed by the user. It provides a kind of relational point of view which is built by the system. The user may add entities to the user-root view or remove them similarly as in regular collection views, but with elements of arbitrary types.

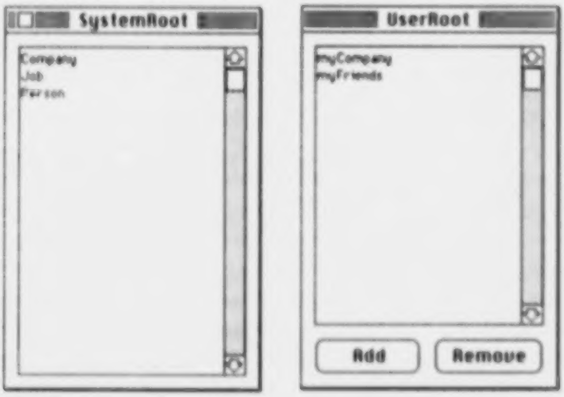
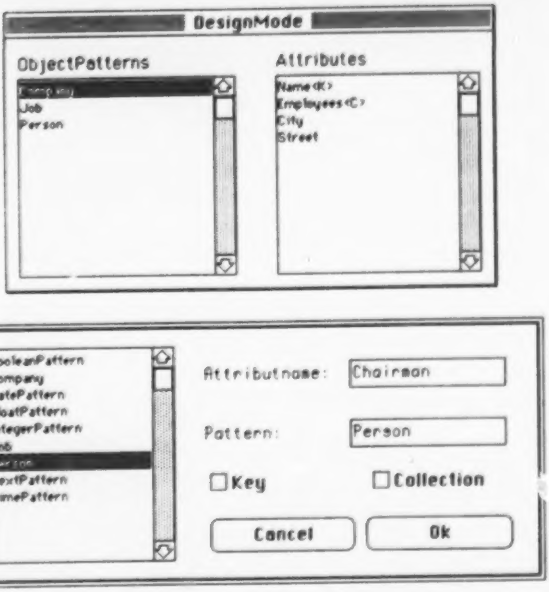

SystemRoot
Company
Job
Person
UserRoot
myCompany
myFriends
Add
Remove

Figure 2. System and user root views


DesignMode
ObjectPatterns
Company
Job
Person
Attributes
Name<K>
Employees<C>
City
Street
BooleanPattern
Company
DatePattern
FloatPattern
IntegerPattern
Job
Person
TextPattern
TimePattern
☐ Key
☐ Collection
Cancel
Ok

Attributname: Chairman

Pattern: Person

Figure 3. Design mode windows

Figure 3 shows the two design mode windows which allow creating, modifiying and deleting attributes and patterns. The windows are rather self-explanatory. The specification of superpatterns is not supported yet in this prototype.

4. Object Structures

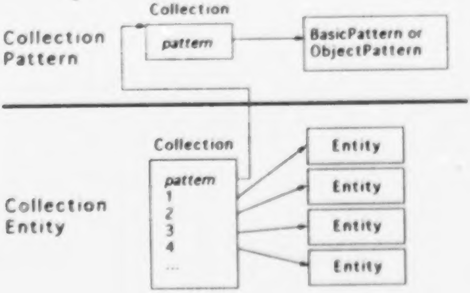

Collection Pattern
Collection
pattern
BasicPattern or ObjectPattern
Collection
pattern
1
2
3
4
...
Entity
Entity
Entity
Entity
Collection Entity

Figure 4. Collection entity and pattern structure

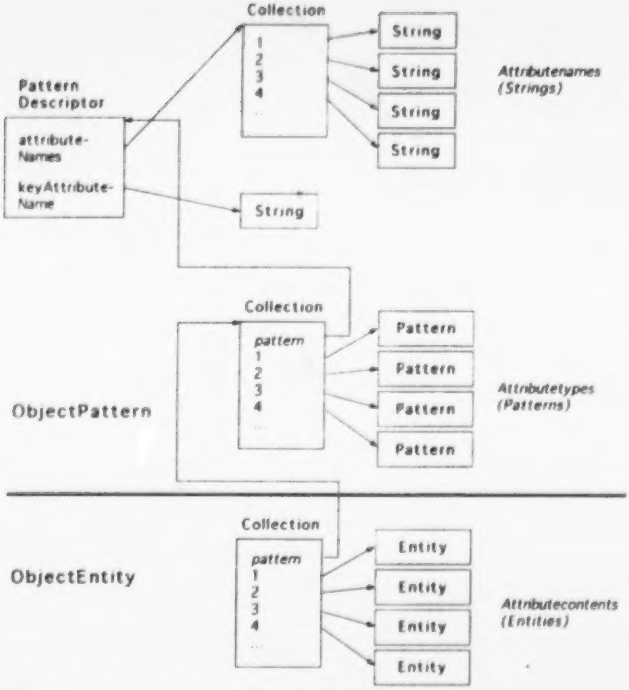

Collection
1
2
3
4
...
String
String
String
String
Attributenames (Strings)
Pattern Descriptor
attribute-Names
keyAttribute-Name
String
Collection
pattern
1
2
3
4
...
Pattern
Pattern
Pattern
Pattern
Attributetypes (Patterns)
ObjectPattern
Collection
pattern
1
2
3
4
...
Entity
Entity
Entity
Entity
Attributecontents (Entities)
ObjectEntity

Figure 5. Object entity and pattern structure

The entity and pattern structures collections of and objects as shown in Figure 4 and 5, respectively, form the basic data model of the current implementation. The patterns are what their name expresses–patterns for their corresponding entities. Pattern attributes are initialized with references to the pattern (type) associated with the attributes. Attributes always hold references of type-conform entities or patterns. Entities and patterns are internally represented by general-purpose, indexable, variable-sized collection objects.

5. Experiences and Future Plans

Although the current prototype provides just very basic functionality, it serves as a vehicle for investigation of new features and for studying integration options with a database system and an extensible interface builder. Our experiments have shown that working with standard views exclusively–which, roughly speaking, show a single entity–leads to a confusing crowding of views. The problem could not be circumvented by heuristic mechanisms like closing the "oldest" view by the system when the number of open views exceeds a certain limit. So the need for user-defined views which can visualize a set of closely related entities at once became obvious. Besides, user-defined views would offer a convenient place for defining and checking integrity conditions. Because object-oriented data models should reflect the most important semantic relations as physical relations, we do not see a neccessity for a powerful query mechanism for information retrival. Instead, we want to offer a small set of general-purpose operations for selecting, searching, sorting, merging, etc. of entities.

We are now working on porting the system from the Apple Macintosh computer to a Windows PC using Smalltalk/V [Digitalk92] as implementation base. The next steps will be the integration of an inheritance mechanism and the incorporation of an extensible interface builder. Last but not least, we plan to connect HyperObjects to a database system to be able to store entities quickly in a safe, access-controlled place.

Thanks to Andreas Obermueller, who implemented most of the first HyperObjects prototype.

6. References

[Rumbaugh91]	Rumbaugh J. et al.: Object-Oriented Modelling and Design, Prentice-Hall, Englewood Cliffs, 1991
[Kim89]	Kim F., Lochovsky F. H., (Eds.): Object-Oriented Concepts, Databases and Applications, ACM Press Addison-Wesley, New York 1989
[Rechenberg90]	Rechenberg P.: Programming Languages as Thought Models, Structured Programming, Vol 11, Nr 3, Springer, New York, 1990
[Atkinson89]	Atkinson M., et al.: Object-Oriented Database Manifesto, DOOD Proceedings, Kyoto Japan, 1989
[Digitalk92]	Digitalk Inc.: Smalltalk/V Windows Product Documenation, Los Angeles, 1992

THE DEVELOPMENT OF A GENERIC REQUIREMENT SPECIFICATION

Kim Trans

Copenhagen Business School, Institute of Computer and Systems Sciences, 1970
Frederiksberg C, Rosenørns allé 31, Denmark

Abstract

When studying software development the general models (Sommerville, 1992) seem easy to implement, but in practice nothing is like in the book. We have at the Copenhagen Business School tried to develop generic software and some problems with the general models have emerged. The problem is mainly how to develop software that can be applied and tested at two user sites and sold afterwards as a commercial product.

The key concept turned out to be generic software, where the actual software were developed as a generalisation of the needs of two involved user companies. This has been accomplished by introducing expert system programmed database interface, control and configuration. The result so far, as the software is still under development, shows that this is a possible method for developing a highly specialised software system that can be sold commercially when the initial project finishes.

The area of concern for the project is to facilitate transparency between shop floor production status and high level management decisions with respect to cost assessment of customised unique products made to order within small and medium sized enterprises. This makes the project a *Computer Integrated Manufacturing* project (CIM) (Waldner, 1992). The resulting system is a decision support module to be used in the assessment and calculation of production costs and in the following pricing of products. The system is to be used for cost assessment and post calculation of the actual cost of a produced item.

1. THE ACTUAL USER CASES

The involved partners, the financial source, and the user companies has been anonymized and for the time being they will be identified as user A and user B.

User A is manufacturing trucks. Recently they had a major consulting firm evaluating their management accounting control system, and an activity based costing model was recommended. This model should increase the awareness of cost in the entire company. Due to the high cost of producing a truck it is essential for user A to be able to do ongoing cost analyses. Cost analyses can help improving production efficiency, and the transparency of cost will be make it easier to identify where cost can be reduced. Cost assessment at user A is presently based on standard cost.

User A needs real time information in order to do dynamic cost assessment on a produced item correctly, to be able to do actual cost calculation on the same item. Actual cost on different levels of the BOM and BOP (Greene, 1987) of any product must thus be provided. Dynamic cost assessment is needed for the precalculation of additional cost emerging from dynamic changes in the production schedule, which may cause extra cost due to rush order of materials, suboptimal production planning, and postponement of work in progress. The actual buyer does not pay this extra cost even though he may cause the additional cost, but when properly documented it can used in future price negotiations.

User B is a department within a bigger company. They are manufacturing parts for cold forging tools. Cost assessment is currently based on tables with standard working hours predefined for parts specified by weight and complexity. This cost assessment method gives a fairly accurate estimate of the overall production cost. The cost of the part is not used directly in price quotations as most prices are based on circumstances relative to the actual buyer. Currently the lower limit on the price is based on the complexity and the weight of the part in question. The upper limit is set by the performance of a corresponding part from another vendor.

This crude pricing scheme causes some problems as not all customers are equally profitable. There are several major cost contributions relative to the profile of the customer, as it might be more expensive to sell identical parts to one customer than to another, depending on the marketing resources used and the physical location of the buyer. User B thus needs a pricing system for the differentiation and classification of buyers, such that any price quotation will cover the involved accumulated production, sales, distribution, marketing and general overhead costs. In addition, when salespersons are visiting potential buyers they need a tool, which can show the financial benefits of buying the product made by user B.

2. THE METHOD(S) APPLIED

The development contract states that the developed software must be useful in the two involved user companies, and sold as a commercial software product to third party when the project finishes. These two constraints create some very specific problems. How is it possible to integrate the requirements of the two actual users with each other and use the emerging requirement specification to develop a commercial cost assessment and pricing system, that can be used by any small and medium sized company that manufactures to order.

The solution method applied is structured in three phases. The first is the derivation of specific requirement specifications in the two user companies. These are then generalised and integrated with elements of general functionality needed in order to do general cost assessment and pricing of unique products. Finally, as the thus specified software is to be used in the two original user companies, it has to be configured to satisfy their specific needs. Thus some kind of set-up program is needed.

2.1 Identification of specific user based requirements using SSM

The first step in the generation of a generic requirement specification is to identify the actual needs of the involved users. To accomplish this a large suite of tools is available (Rosenhead, 1989). We have chosen *Soft Systems Methodology* (SSM) (Checkland, 1990) as it is a very general methodology for the construction of a basic model of the problem domain and the surrounding company context. SSM was chosen based on the result of an initial future workshop, where the application domains of cost assessment and pricing were identified.

SSM involves looking at the actual organisation – finding its main purpose, problem themes and upon this creating a statement concerning what information systems are needed and what they will do. The core purpose using this methodology is to construct and present a model of mutual understanding of the problem situation in question. In the actual case, the presentation of the problem situation is meant to serve as a basis for a discussion of specific requirements. SSM involves the following major activities,

- The construction of a rich picture
- Finding tasks and problematic issues to be solved
- Proposal of relevant systems to solve tasks and issues
- Describing these systems using conceptual models

This chain of activities is iterated until a mutual understanding of the domain has been established. This iteration process has been done as in the standard method, with the exception that the conceptual models has been omitted. This is due to the fact that the emerging result is not to be used to design appropriate systems, but as the foundation of the following generalisation process, where the final generic system will be specified. Thus only our changes to the standard definition of a conceptual model (Rosenhead, 1989) will be described. A standard conceptual model is,

- The first draft of the system design
- A medium of communication to select correct designs
- The basis of a requirement specification

The conceptual model, contains the set of activities and the logical relation that must exist between them in order to make the desired transformation specified by the actual relevant system. An important part of this process are the activities performed by a control system to measure performance of the other activities. All these specific conceptual activities has been postponed until after the integration of general domain knowledge, in order to specify a generic system that can be sold commercially. Until now everything is fine, but how can the result of the two user SSM's be generalised in order to define a solid foundation for a commercial product.

2.2 Generalisation of the user specific requirements.

In order to generalise the results from the two user SSM's, just take the union of the identified user requirements. This seems very easy, but problems arise when this approach is used without care. The problem is that if the two user studies together do not give the total set of necessary requirements, it will be impossible to use the generalised specification, thus it will be inadequate as a specification of a commercial product.

The obvious way to make sure that an appropriate specification is made, is to put the developed specifications aside for a moment, and try to identify what a corresponding commercial product specification should be. In this case several possibilities to continue the specification development process are open.

Firstly, two user cases might be to few - a much larger number of user studies seem more appropriate. The thus emerging problem is when to stop this process, and of course the additional cost of doing many SSM's in different user companies. The first problem can be solved by carefully selecting the case studies such that they all contribute to the final result and still try to keep the number of cases down to a minimum (Holland et al, 1986). The money problem is happily out of scope of this paper, so it is somebody else's problem.

Secondly, domain knowledge collection from relevant literature can tell a lot about actual and potential users and their corresponding needs. This can be done as pure technical knowledge collection, pure user group and their corresponding needs knowledge collection, or maybe something else. The main objective is that it can be done without interaction with users, thus eliminating travelling costs, but still be able to identify potential users and their probably even more specific needs. Probably no user within any actual context can be characterised as an average user.

If these two methods are integrated and then used iteratively, they can in fact solve the problem of specifying a generic product. The iterative cycle can be described in the following way.

> continue domain knowledge collection followed by user analyses in small steps, until a satisfying specification has been developed or you run out of money.

In the actual case this approach has been followed and a suitable specification has been developed. No serious problems this far, but how do we then fulfil the needs of the original two users. Their needs is of course satisfied, but in a very general way as the system by now is heavily based on commercial interests. This means that the developed requirement specification, at this point does not directly support the specific needs of the initial two users, but the support of the two original users is part of the development contract and thus has to be accomplished. The problem is then, how is it possible to customise an advanced piece of software to fit into the actual context of a specific user.

2.3 Expert system based configuration of a generic software system

In order to dynamically configure highly specialised software systems it is necessary to program them in some way. This can be done using a manual configuration supported by a decision tree, or it can be done with meta-programming (Reeve, 1988). In the present case the fundamental configuration can be done with mouse-clicking on the screen, but this process is very time consuming if a lot of details are to be specified. In order to do efficient meta-programming, an expert system based interface (Rich and Knight, 1991) is supported. This interface is automatically programmed in case of a standard configuration of the developed system, but it is possible to individually program the specified software system to do whatever wanted, using an appropriate rule base.

In order to implement a rule based expert system configuration interface, three different functional types of rules are needed,

- Input specification rules,
- Process specification rules,
- Output specification rules,

where *input specification rules* define where input data are stored in the host system, and map these data to the internal database tables. *Process specification rules* specify what kind of processing is to be done with the input in case of a non-supported model or method being applied, or just to fit actual values to actual context dependent calculations. Finally *output specification rules* define what to do with the obtained results, i.e. either to store them in the company database by transforming data to the right format, to write database reports, or to display the data directly to the user.

This approach have two fundamental flaws. Knowledge acquisition has to be done in order to configure the developed system and some kind of knowledge maintenance tool has to be available. The first problem is a general phase in the development of expert systems (Bramer, 1990), and thus a well-known problem that can be treated appropriately.

The second problem is more serious as it can cause an already existing and appropriate knowledge base to be seriously wrecked. The problem is a consequence of the fact that additional rules can destroy an already existing functionality (Trans, 1994). The only serious candidate for solving this problem is to generate a logical model of the entire target domain, and then use this model to evaluate whether the introduction of new rules will conflict with it. This of course is time consuming, but the question cannot be solved in any other way as it demands a semantic validity check, which is impossible due to Gödel (Hamilton, 1978), but heuristic checks are still possible.

3. USE, EXPERIENCES AND RESULTS

Here follows a description of the experience gained using the above described three phase method to develop a generic requirement specification. The description follows the phases of the method closely.

An initial future workshop was used to identify the actual domains of application. This identification is needed in order to involve the right developers and domain experts, as it otherwise would be difficult to gain sufficiently insight into the domain. The results were that user A wants information on estimations and actual costs of production in relation to each item produced and user B wants sales support in order to do realistic quotations differentiated on available customer information. The question is then, whether there is any basis for generalisation.

3.1 Multiview on user sites

Based on information from user interviews, rich pictures were constructed together with the actual user participants. The essential results of the SSM process were,

Major points of discussion,

User A

- Accurate cost assessment
- Inavailable information
- Problem spots in production
- Attention to human aspects

User B

- Exact real time costing
- Overhead cost on each product
- Price differentiation
- Difference in knowledge

Tasks and issues,

User A

- Improvement of quality
- General cost assessment
- Activity based costing
- World view on data
- Customer relations

User B

- Redundancy in tables
- Maintenance of pricing program
- Relevance of the costing module
- Relevant examples to the customers
- First time lifetime improvements

Relevant systems,

User A

- Cost assessment cycle
- A cost assessment system

Cost assessment cycle:

The system transforms a specification of a product or prototype to a cost assessment or transforms poor estimates to better estimates. The world view is, *the calculation process is iterative*.

Cost assessment system:

The system transforms a specification of a truck to an estimated cost of the same truck. The world view is, *we need better and earlier cost estimates, and cost assessment is possible*.

User B

- Production cost assessment system
- Pricing and quotation system

Production Cost Assessment System:

The transformation consists of transforming the specification, weight and corrective factors of a part to a cost assessment of the part. The world view is, *cost assessment is possible*.

Pricing and quotation system:

The transformation consists of transforming a specification and the production cost of a part to a quotation price. The world view is, *we can only sell if we offer the right market price to the customer*.

3.2 Generalisation

The information described above from the SSM process seems easy to integrate to a unified solution, but what about the general case. This problem arises, as the final system will be sold commercially and thus be used by the average order producing company. The generalisation of the user requirements is done in two steps, where the first generalises the specific needs of the users, and the second step incorporates general functionality necessary to specify a commercial product. This section ends with a short description of the relation between the derived specification and the specific needs of the two users.

The two user A relevant systems, show that the most important tasks to be solved are mainly related to cost assessment. The two user B relevant systems, show that the most important tasks to be solved are mainly related to real time cost assessment and pricing. A generalised specification can thus be centralised on the domain of costing for both users and additionally on pricing for user B.

A general costing system consists of a number of methods to aggregate cost data and relate the same to the actual production data. A costing system should support one or more of the following tasks,

- Documentation
- Stock taking
- Management support
- Ad hoc analyses

The strategic aspects of costing systems can be classified through their use of cost allocation strategies. Examples of cost allocation strategies are,

- Activity based costing
- Full costing
- Contribution costing

With the above mentioned four purposes and three cost allocation strategies in mind, it is now possible to define a general cost assessment and pricing system supporting both the two involved user companies and commercial exploitation. The specified system *COST* has three major functional elements,

Strategic costing and pricing

The strategic and tactical function is used to tailor the costing and pricing functions, with respect to the costing and pricing strategies applied in the actual user company. The strategic and tactical user view, presents the user with a dialogue of choice points concerning company characteristics, product domain, and style of competition. The answers to these questions lead to a number of system generated suggestions for proper utilisation of the costing and pricing methods as a general strategy. When costing and pricing strategies are decided upon, the strategic and tactical view offers support to maintain the chosen strategies through various tactical modifications of the used data.

Costing

Assessment and calculation of production costs are performed with different user tasks in mind. A specific user task is accomplished by activating an appropriate scenario, and based on actual production data cost assessments are made in correspondence with the defined strategies for costing.

Pricing

A pricing system is a natural extension of a CIM company costing system. Pricing presents a price view on the basis of scenario driven production costs. The pricing view is constructed in correspondence with the established pricing strategy. Very often different customers are willing to pay different prices for the same product. This means a higher profit can be achieved, if the company can charge different prices according to the customers willingness to pay. The price strategy consists of dimensions for which price differentiation can be accomplished (Kottler, 1988).

Costing at user A is divided into three levels where each level specifies certain actions to be performed. The strategic costing level is important for user A as they expressed a need for doing cost allocation differently, maybe using activity based costing. Tactical costing is necessary for user A in order to specify details in the cost allocation method. Operational costing is essential for to doing cost assessment on specific products and orders.

Real time cost assessment is part of the operational costing at user B. Pricing is divided into three levels depending on who can decide in the different pricing situations. Operational pricing serves the needs of sales personal and for it to be carried out efficiently, historical information on earlier pricing decisions must be available.

3.3 Expert system based configuration for the two users

In order to apply the above defined expert system based configuration of the generic COST software system, rules are needed. These rules are needed in order to make the developed generic software able to function independently of other systems as integration is a main objective. This means that a general CIM environment model (Kusiak, 1990) has to be defined, which must include all data needed in any possible configuration of the COST system. In relation to this, a set of general rules for the three cases of input, processing and output will be defined, and additional context specific rule bases for the two users will be provided. The inference machine applied is a standard forward chaining inference machine (Rich and Knight, 1991), as the one defined in (Trans, 1991). The actual implementation of it is called the rule handler.

Input specification

The fitting of the user company database in order to set the current context, is thus the process of relating the general CIM model applied in COST to the actual user company context. This is mainly done via the internal database dictionary but can be supported by rules, which will be interpreted by the rule handler. Thus the rule handler takes care of the more specific constraints, and helps establishing the general CIM interface, where the company CIM environment in question is mapped to the internal COST database tables. In this way I/O can be done as simple data transformations between the company database and the general CIM environment model internal in COST. Where the obtained result is used to map user company databases to COST tables, the results of the input mapping is written directly into the COST dictionary tables.

The same rule handler is used in other parts of COST where rules are applied. The rule handler then interprets the active rules, and in this way programs COST in order to conform to the specified functionality based on the original files in the user company database. The rule base thus contains an additional program given as a set of potential corrections and the rule handler functions as a rule interpreter which corrects the actual data.

Process specification

Program configuration and the correction of actual data using the rule handler can be performed in two modes. This includes the customisation of database links ,where rules defining constraints on the mapping between external and internal tables can be executed by the rule handler as SQL meta rules (Date, 1990). In all these cases, the used rules define knowledge specific for the actual user company and the actual situation of use.

The internal application table, field and value mapping, where the present customised configuration is defined, is done in order to process data in a company specific way, as special constraints will hold for each company using COST.

Processing is done by applying the rule base to the data stored in the COST database. The result of the inference process is written back into the COST database. Process rules, specify pre-processing to be done when a pricing or cost calculation on actual values is done. This is where the specific knowledge regarding the actual user organisation is used, to transform stored data into actual contextual values for the calculation in question.

Output specification

Finally the rule handler is used to specify formats for user screen and database report output, either defined in an appropriate form for a report generation tool or for it to be written back into the COST database. Database output is written directly to the database table in question and if reports are required they are made. The most general feature of rule based output specification, is that it is possible to let the rule handler generate SQL code in order to make user specific views against the internal COST database.

The first two kinds of output is quite simple, as they are plain database I/O, but the third kind has to be further elaborated. The idea is to let the rule base use meta rules, in such a way that it will generate a sequentially ordered sequence of symbols, which then can be fed to and interpreted by an SQL DBMS. This approach is further described in (Trans, 1992).

3.4 User specific expert system based configuration

The above specified abstract processing of the rule base is used to generate the following functionality for the two users.

- Specify relations between actual cost elements, system variables, and applications
- User database to COST database table groupings
- Setting constraints on user dependent system variables

This functionality has to be programmed, and it will include knowledge acquisition in the two user organisations. The thus acquired user company specific knowledge has to be restructured as rules before it can be used to program COST.

4. CONCLUSION

The conclusion of this work in progress - we are at the end of the design phase, and at the beginning of the prototype and development phases - is that our approach to the development of a generic system works quite well. We feel that we do know the needs of the involved users, that we are able to generalise those needs to a commercial software system, and that we can retro-fit the emerging system to all potential users in the commercial target group, including the two original users A and B. The initial goal has clearly been obtained as a suitable requirement specification has been made.

5. REFERENCES

Bramer, M., (1990), "Practical Experience in Building Expert Systems", Wiley, Chichester.

Checkland, P., and Scholes, J., (1990) Soft Systems Methodology in Action, Wiley, Chichester.

Date, C.J., (1990), "An Introduction to Database Systems 5.ed, Addison-Wesley, Reading.

Groover, M.P, (1987) "Automation, Production Systems, and Computer Integrated Manufacturing", Prentice-Hall, Englewood Cliffs.

Hamilton, A.G., (1978), "Logic for Mathematicians", Cambridge University Press, Cambridge.

Holland, J.H., Holyoak, K.J., Nisbett, R.E. and Thagard, P.R., (1986), "Induction", MIT, Cambridge.

Kottler, P., (1988), "Marketing Management - Analysis, Planning, Implementation and Control 5.ed", Prentice-Hall, Englewood Cliffs.

Kusiak, A., (1990), "Intelligent Manufacturing Systems", Prentice-Hall, Englewood Cliffs.

Rich, E. and Knight, K. (1991) "Artificial Intelligence 2.ed", McGraw-Hill, New York.

Rosenhead, J., (1989), "Rational Analysis for a Problematic World", Wiley, Chichester.

Rowe, N.C., (1988), "Artificial Intelligence through Prolog", Prentice-Hall, Englewood Cliffs.

Sommerville, I., (1992), "Software Engineering 4.ed", Addison-Wesley, Wokingham.

Trans, K., (1991), An Expert System Shell for Planning with Automatic Feedback Learning, *in:* "Scandinavian Conference on Artificial Intelligence -91", B. Mayoh, ed., IOS Press, Amsterdam.

Trans, K., (1992), A Simple Natural Language Interface, *in:* "NordDATA'92 Preedings", J. Leponiemi, ed., PITKY, Tampere.

Trans, K., (1994), Autopoetic Restructuring of Knowledge Bases, *in:* "Advances in Artificial Intelligence -Theory and Application II", J.W. Brahan and G.E. Lasker, ed., The International Institute for Advanced Studies in Systems Research and Cybernetics, Windsor

Waldner, J., (1992) "CIM - Principles of Computer Integrated Manufacturing", Wiley, Chichester.

AN OBJECT ORIENTED TECHNIQUE FOR SYSTEMS SPECIFICATION

Jesús Torres, José A. Troyano and Miguel Toro

Dpto. de Lenguajes y Sistemas Informáticos, Facultad de Informática y Estadística, Universidad de Sevilla,
Avd. Reina Mercedes s/n, 41012 Sevilla, Spain

Abstract

In this paper we show the object-oriented language for systems specification called TESORO. In this language an object consists of a state, a behaviour and a set of transition rules. We construct a system model as a parallel composition of interacting objects of several classes. These classes can be simple if we describe their features explicitly, or complex if they are defined over other classes. Furthermore we show two distinct specification styles for the objects behaviour.

1. INTRODUCTION

It seems to be easy enough that the best way to study a complex system is to decompose it in smaller subsystems. This philosophy can be extended to software development. However, it is not so clear the way in which this decomposition is carried out. In structured methods, traditionally used, the problem model and the software model that solves the problem are so different.

A natural way to understand a system is considering it as a set of cooperating objects that are working to reach a common goal. In this sense, there are development techniques, as the object-oriented design (Rumbaugh et al., 1991); in these metodologies the decomposition is made using the structure offered by the system components and the relationships among them. In this way we have a software model (called object-oriented model) related to the problem model.

We have just seen a natural way to understand a system, now we need a notation clear and concise enough that allows us to describe the system model without ambiguities (Jungclaus et al., 1991). An effort to describe clearly the system model in the early phases of software development (analysis and design), will make the last stages of the software life cycle easier (implementation, proof and maintenance). As a result of this we can use a formal specification language as a tool in order to describe the system model. This formal language allows us to describe the model unambiguously and can help us to verify some properties of the model.

The fundamental constructs in the object-oriented model are the objects. An object consists of a set of attributes that characterizes the state of the object, a set of events and processes that describes the behaviour of the object and a set of transition rules which denotes the change in the state of the object before the ocurrence of an event. The objects

534

that share these characteristics are grouped in classes, which are the basic units in our specification language. We can also describe relationships among classes that determine which form can interact objects of the classes related.

To describe the object attributes and the operations defined over them (transition rules) we use algebraic data types. To specify the object behaviour we use an algebraic model of process description. In this way we describe the object behaviour as the possible trace of events in his life (the period of time between the events *create* and *destroy*). In this sense is interesting the paper (Torres et al., 1993), where we present a relationship between an object-oriented specification language and the formal description technique LOTOS (ISO, 1988), which integrates the algebraic specification language ACT ONE (Ehrig and Mahr, 1985) for the description of data structures and a process description language, based on CCS (Milner, 1980) and CSP (Hoare, 1985) for the description of behaviour. In this way we describe the object model using an algebraic specification language that can help in the task of verifying properties.

The organization of this paper is that follows. This introduction constitutes the first section. In second section are presented the features of the object-oriented model. Third section describes the general structure of a specification in TESORO. In fourth section simple classes are presented. Fifth section presents algebraic abstract data types as a way for representing object attributes. In sixth section we describe the relationships among classes. Seventh section presents complex classes. In eighth section we illustrate the descriptive power of TESORO by means of dining philosophers problem. In nineth section we extract conclussions and expound future work.

2. THE OBJECT-ORIENTED MODEL

When we make a system model, we can get the benefit of the structure imposed by the system. The components of a system are interrelated and are interdependent; a set of independent components does not make up a system. The main task in modeling the system will be to identify the components and to determine the relationships among them. Every component can be represented by means of an object.

The object-oriented concept has its origins in the object-oriented programming, which sees the programs as a set of interacting objects that have a state and offer a functional interface by methods. This notion is used also in the analysis and design phases of a computer proyect.

The main features of the object-oriented model are (Booch, 1991): (1) abstraction, that is a simplified description of the system which only insists on details which are outstanding, (2) hiding information, that is the process to hide the details of an object which do not contribute to its main features, (3) classification, that groups into classes objects which share common features, (4) hierarchy, that is an abstraction ordering provided by inheritance, (5) concurrence, that describes the execution of cooperating processes which synchronize and communicate among them, and (5) identification, that serves to reference an object uniquely during all its life.

2.1 Object-Oriented Model Construction

The object-oriented model is described through a set of classes and relationships among classes specification. A class defined in a specification can be simple or complex.

- Simple Classes. These classes are described without making reference to other classes, and they specify the structure and behaviour which shares a set of objects.

An object is compounded of a state, which is characterized by a set of attributes, a behaviour and an interaction with the environment, which are described by means of events and processes, and a set of transition rules which denotes the changes of states.

- Relationships. In the model we can define relationships among classes. These relationships are based on the synchronization and communication of the objects of several classes, which is achieved by shared events.
- Complex Classes. The complex classes are defined from other classes with the next constructors:
  1. Inheritance. One feature which is not defined with the simple classes and the relationships is the hierarchy of abstractions. This is achieved with the inheritance, where a new class is defined from one class (simple inheritance) or several classes (multiple inheritance). The new class is called son class, the existing classes are called father classes, then the son class inherit features from the father classes. In addition, we can append new features (called emergent features) to the new class.
  2. Aggregation. It defines a new class based on the relationships of existing classes and several emergent features.

2.2 The role of Abstract Data Types

In order to describe the attributes and transitions we need some data types and operations over them. The classes are built on these data types, which serve to define the object identification and state domain. With the idea to give a formal definition for data types, we are going to use an algebraic specification sublanguage.

In next sections, we describe TESORO, an object-oriented technique for systems specification.

3. SPECIFICATION

The specification in TESORO is composed by three sections:

- Library. In this section are enumerated the abstract data types that are used in the rest of specification.
- Classes. Here we define the classes that will appear in the specification.
- Relationships. As we have said above, the classes in a specification are not independent among them. So, in this section are described the relationships among classes which compound the model.

Specification syntax is the following:

```
Specification <specification name>
   Library <abstract data types used>
   <classes specification>
   <relationships specification>
End specification
```

4. SIMPLE CLASSES

For every class, we describe the structure and behaviour of the set of objects that it represents.

The class specification is composed by three sections:

- The attributes section describes the structural aspects of a class. Here are defined the attributes that we use for object identification, the constant attributes, whose values do not change during all the object life, and the variable attributes which make up the object state. Every attribute has a type. This type can be an abstract data type or even an object type, making possible to refer an object with its identification. Furthermore, we can impose a set of static constraints over the value attributes, so these constraints can never be broken.
- The events section describes the behavioural aspects of a class. The events can be internal to the system, or external if they denote an interaction with the environment. We can define parameters associated to an event. These parameters let us communicate data among objects when an event occurs. The parameters may be send or receive depending on the communication way. There are two special events, one which denotes the object creation (create), this is, the way we have to introduce a new object in the system, and other which denotes the object destruction (destroy), this is, the way we have to eliminate an existing object of the system.

The object behaviour is specified by means of permissions and triggers, which are boolean expressions. With permissions we say when an event can ocurr, and with triggers we represent the object responses when it is found in a certain state. The dynamic constraints impose an event order, which is described by means of processes specification. This specification is made up using a subset of process algebra constructs (Hoare, 1985; Milner, 1980). These constructs are the operator ; (prefix action), the operator [] (choice composition), the operator ||| (interleaving composition) and the recursive processes description.

- The transitions section describes how to change the variable attributes values of the object in a class (and in consequence its state), by means of events occurrence or by means of changes of other attributes.

Depending of the shape in which the attributes change their values, we can classify them in derived attributes, which are variable attributes whose values depend on others attributes, and not derived attributes, which whether are constant or identification attributes, or variable attributes, which value is modified when a certain event ocurrs.

Syntax of the simple classes specification is the following:

```
Class <class name>
  attributes
      identification
          <attribute name>:<type>; ...
      constant
          <attribute name>:<type>; ...
      variable
          <attribute name>:<type> {(<inicialization>)}; ...
      static constraints
          <condition>; ...
   events
      external
          <event name>{(<formal parameters>)}; ...
      internal
          <event name>{(<formal parameters>)}; ...
      permissions
          [<condition>] <event>{(<actual parameters>)}; ...
      triggers
          [<condition>] <event>; ...
      dynamic constraints
          <processes descriptions> ...
   transitions
       from events
           <event>{(<formal parameters>)}
                                    -> <attribute> = <expression>; ...
       from attributes
           <atribute> = <expression>; ...
 End class <class name>
```

The specification language TESORO supports two specification styles for objects behaviour:

- Constraint Oriented Specification. This style is characterized by the use of the process algebra operators to specify the object behaviour as the valid events traces set in the object life. We use dynamic constraints and we do not make explicit reference to object internal state.
- State Oriented Specification. In this style we define a set of variables which make explicit the object state all the time, describing the behaviour by means of a set of events ocurrence permissions. Then from a certain state and applying a set of transitions we can determine the state changes after an event ocurrence.

5. ABSTRACT DATA TYPES

The classes (in particular the attributes) are defined over domains. These domains are, in fact, abstract data types (ADT), and they consists of a sets of data values and a set of operations over these values. We use algebraic data specification to describe these domains.

When we write a specification, we must define the ADT's necessary for the definition of object attributes. For example we can use generic types in order to group more basic ADT's with the well-known collection mechanisms (stacks, sequences, queues, sets, maps, etc.).

The language used for describe ADT's will be ACT ONE. In this language data specifications are collected into *type* constructions. A type consists of a set of sorts which represents the possible sets of values, a set of *operations* which describes the signature of the type functions, and a set of *equations* written as equalities of expressions of the type.

Provided that we use ACT ONE for the abstract data type specifications, we do not describe here the syntax of this language. The interested readers are refered to the bibliography (Ehrig and Mahr, 1985).

The abstract data types used in a specification, are included into the *Library* section, for example:

```
    Library Boolean, Names, Positions
```

6. RELATIONSHIPS AMONG CLASSES

The relationships connect objects through the syncronization of their events. These relationships allow us to describe the bonds among the separate components of the system.

When we establish a relationship, we make possible that objects of related classes share the events involved in the relationship. We can designate this events with a different name for each class, but in fact this is only a syntactic facility, because all the events of objects of different classes related by a relationship represent the same event.

To specify the relationships among classes we are going to use the following syntax, on the one hand we enumerate the variables and variable types used in the expressions for the events parameter or the objects identification, on the other we enumerate the bonds which establish the communication channels among the objects of related classes.

```
Relationship <relationship name> among <class1>, <class2> ...
   [for all
       <variable name>:<type>; ...]
   bonds
       <class1>(<id_ob1>).<event1>(<parameters1>) =
       <class2>(<id_ob2>).<event2>(<parameters2>) ...;
                              ...
End relationship <relationship name>
```

7. COMPLEX CLASSES

Till now, the only available mechanims to describe a system model are the simple classes and the relationships among classes. At certain cases these mechanisms are not enough for describing all the features of a system. For this reason, we introduce the complex classes as a new resource to describe a system model. The complex classes, are defined over other classes with the inheritance and aggregation constructs.

7.1 Inheritance

The inheritance is a powerful abstraction that allows us to define a new class of objects as an extension of existing classes. The new class inherits the structural and behavioural aspects of the other classes. Besides the inherit features, we can define emergent characteristics for the new class.

Associated to the concept of inheritance, appears the modificability, this is, the capability that the son class has to alter the characteristics of the father classes. In this sense, and accepting the classification proposed in (Wegner, 1990) the inheritance available in our language is at the same level that behaviour compatibility. In this manner we only can impose stronger constraints (through the sections *static constraints*, *dynamic constraints* and *permissions*) to make the behaviour of the son class compatible with the behaviour of the father class.

As we have already commented, the inheritance can be simple or multiple. In the simple inheritance we have a specialization of the father class. This specialization can be temporary or permanent. We have a temporary specialization if the events create and destroy of the son class are different of the father class ones. In the permanent specialization the events create and destroy are the same for the son and father classes, so the life of an object of the son class is always bound to the corresponding object of the father class.

Multiple inheritance appears when a son class has more than one father classes. In this case, the son class has its own events create and destroy and the lives of its objects are not bound to the objects of fathers classes.

Specification syntax of the simple inheritance is the following:

```
Class <class name> inherits from <father class>
               [where <condition>]
      [attributes     ...]
      [events         ...]
```

```
    [transitions    ...]
End class <class name>
```

The *where* clause, which is a predicate over the constant attributes, indicates the class which the objects we create belong to. In the *attributes*, *events* and *transitions* sections are described the emergent properties of the new class.

Specification syntax of the complex inheritance is the following:

```
Class <class name> inherits from
                 <father class1>, <father class2> ...
    [attributes     ...]
    [events         ...]
    [transitions    ...]
End class <class name>
```

Like simple inheritance, in the attributes, events and transitions sections we describe the emergent properties of the new class.

7.2 Aggregation

The aggregation of classes is based on a similar concept that we used in the relationships among classes, because establishes connections among objects by means of its synchronization through events. However, the aggregation gives class features to a relationship, so we can add attributes and behaviour to the aggregate class.

Every bond that is defined in the relationship over which is defined the aggregate class, is matched with an event of the new class. One of these events must be the create event and another must be the destroy event (if it exists) of the aggregate class.

Aggregation syntax is the following:

```
Class <class name> aggregates <class1>, <class2> ...
    [attributes     ...]
    [events         ...]
    [transitions    ...]
    relationships
       [for all
          <variable name>:<type>; ...]
       bonds
          <class1>(<id_ob1>).<event1>(<parameters1>) =
          <class2>(<id_ob2>).<event2>(<parameters2>) ...;
                               ...
End class <class name>
```

In the relationships section is specified the bonds among classes over which is defined the aggregate class.

8. EXAMPLE

The next example, which let us show the simple classes definition, is the dining philosophers problem (which is a typical problem in the concurrent programming). The problem description is the following: Five philosophers sit around a circular table. Each philosopher spends his life alternately thinking and eating. In the centre of the table is a large platter of spaghetti. Because the spaghetti is long and tangled (and the philosophers are not mechanically adept), a philosopher must use two forks to eat a helping. Unfortunately, the philosophers can only afford five forks. One fork is placed between each pair of philosophers, and they agree that everyone will use only the forks to the immediate left and right.

To specify the dining philosophers problem, we are going to define the Philosopher and Fork classes:

```
Class Philosopher
   attributes
      identification
         name: Name;
   events
      external
         birth(create);
         death(destroy);
         think;
         eat;
      internal
         take(send f:Fork);
         release(send f:Fork);
      dynamic constraints
         process phil_life :=
            think;
            (take(left(name))
                     |||
            take(right(name)));
            eat;
            (release(left(name))
                     |||
            release(right(name)));
            phil_life;
         end process
End class Philosopher

Class Fork
   attributes
      identification
         number: Position;
      variable
         available: bool(true);
   events
      external
         put(create);
         remove(destroy);
      internal
         in_hand;
         in_table;
      permissions
         [available] -> in_hand;
         [not(available)] -> in_table;
   transitions
      from events
         in_hand -> available = false;
         in_table -> available = true;
End class Fork
```

In this example, the class Philosopher uses the type Names, and the class Fork uses the type Position. Now we show the algebraic specification of both types:

```
type Names
   sort Name
   opns
      Susana, Jose, Maria, Carmen, Andres: -> Name
endtype Names

type Positions is Names
   sort Position
   opns
      1,2,3,4,5: -> Position
      left, right: Name -> Position
   eqns
      right(Susana) = 1;
      right(Jose) = 2;
      right(Maria) = 3;
      right(Carmen) = 4;
      right(Andres) = 5;
      left(Susana) = 5;
      left(Jose) = 1;
      left(Maria) = 2;
      left(Carmen) = 3;
      left(Andres) = 4;
endtype Positions
```

We need to establish a relationship between the Philosopher and Fork classes. This relationship shows the fact that when a philosopher takes a fork it must dissapear from the table, and when a philosopher releases a fork it must be available in the table again.

```
Relationship Philosopher_fork among Philosopher, Fork
   for all
      p : Philosopher;
      f : Fork;
   bonds
      Philosopher(p).take(f) = Fork(f).in_hand;
      Philosopher(p).release(f) = Fork(f).in_table;
End relationship Philosopher_fork
```

In this relationship, we use the variable f, which has the same type of a fork identification, to identify the fork that is taken or released.

In this example we can see the two specification styles. In the Philosopher class, we use the dynamic constraints to describe the class behaviour as the possible sequences of events. However, in the Fork class, we have defined the class behaviour by means of a transition system in which two states are defined for a fork, available and not available, and the transitions from one to another. In TESORO, it is allowed to combine both specification styles, this let us extend the language expressive capacity.

9. CONCLUSIONS AND FUTURE WORK

We have shown an object-oriented specification language. With this language we describe an object-oriented model whose principal constructs are the objects. We have a vision of an object that consists of three fundamental parts, the structure imposed by its attributes, the behaviour described by the possible sequence of events and his funcionality defined by a set of transition rules. All the objects that share the same characteristics are grouped into classes. We also allow to describe relationships among objects of distinct classes. With these features, we consider the system model as the parallel composition of objects. We have shown the algebraic aspects of our specification language, useful for verifying properties. We also have presented two distinct specification styles for the objects behaviour, showing two approachs, one in a more declarative sense and another one in a more operational sense.

The future work is going to be organized in order to 1) specify an operational semantic for our language, based on a basic transition system (Manna and Pnueli, 1992), 2) generate a prototype from the specification, 3) verify properties of a model and choose a notation to specify these properties. At present we also are developing graphical tools that will constitute a work environment for the analysis and design phases in software development.

10. REFERENCES

Booch, G., (1991), "Object-Oriented Design with Applications", Benjamin Cummings, Redwood City, California.

Ehrig, H. and Mahr, B., (1985), "Fundamentals of Algebraic Specification, Part 1" Springer Verlag, Berlin.

Hoare, C.A.R., (1985), "Communicating Sequential Processes", Prentice-Hall International Series in Computer Science, New Jersey.

ISO, (1988), "LOTOS, A Formal Description Technique based on the Temporal Ordering of Observational Behaviour", ISO-Information Processing Systems - Open Systems Interconnection ISO 8807.

Jungclaus, R., Saake, G., Hartmann, T. and Sernadas, C., (1991), "Object-oriented specification of information systems: the TROLL language".

Manna, Z. and Pnueli, A., (1992), "The Temporal Logic of Reactive and Concurrent Systems. Specification", Springer-Verlag, New York.

Milner, R., (1980), "A Calculus of Communication Systems", LNCS, Vol. 92, Springer-Verlag.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., (1991), "Object-Oriented Modelling and Design", Prentice-Hall, New Jersey.

Torres, J., Troyano, J.A. and Toro, M., (1993), "Desde el lenguaje de especificación orientado a objetos TESORO a LOTOS", Preliminary accepted to the *Informática y Automática* Journal.

Wegner, P., (1990), "Concept and paradigms of object-oriented programming", OOPS Messenger, ACM Press, Volume 1, Number 1.

Education

AN ENVIRONMENT FOR RESEARCH AND EDUCATION IN INFORMATION SYSTEMS

Rommert J. Casimir

Tilburg University, P.O. Box 90153, 5000LE Tilburg, The Netherlands.

Abstract

Experiments are not extensively used for research in information systems design. One of the reasons for this state of affairs is the lack of suitable tools. To remedy this shortcoming, a management game simulating detailed transactions was built. It was used in a number of courses. Preliminary results and research and education potential are discussed.

1 INTRODUCTION

An important field in the discipline of information systems is the construction of methods for information systems design. Research strategies to determine whether such methods really accomplish better information can be classified into ***empirical research*** and other types of research. Empirical research can be classified into *case studies, field studies, field tests* and *laboratory tests* (Van Horn, 1973; Hamilton and Ives, 1982). In a field test, the same real life information system is designed with and without the method under consideration. In a field study, data are collected on information systems designed with and without application of the method under study, and statistical techniques are used to compute the influence of that attribute. In a laboratory test, the information system is designed and implemented in a contrived environment with and without applying the method to be studied. In the realm of nonempirical research, a distinction has been made between *theorem proof, engineering* and *subjective argumentative* (Vogel and Wetherbe, 1984). Theorem proofs rely on strictly formal reasoning, whereas subjective argumentation relies on informal reasoning. For example, in the newsboy problem, (Eppen et al., 1987; Lapin, 1988; Shogun, 1988), it may be either formally or informally proven that a newsboy with perfect information will attain higher profits than a newsboy with imperfect information. No serious operations researcher will think it necessary to corroborate such results by interviewing newsboys in the field.

Formal proofs can only be used for those stages of information systems design where transformations are strictly formal. For example, it can be formally proven that a program in an applicative language corresponds to the specifications in a formal specification language, but it cannot be formally proven that the formal specifications in the specification language correspond to the informal specifications a user has in mind. Though most practical system designers think the latter problem is most important, the notion that formal transformations are the primary element of software engineering has been strongly defended (Gries, 1991).

Both field tests and field studies suffer from high cost. In a field test, design costs are doubled because the information system under study must be designed at least twice. The cost of a field study is not high in it self, but the risk of applying an unorthodox method is so high that it will often be avoided, even it has a reasonable chance of success.

Considering the disadvantages of other research strategies, experimental research seems a promising field, but it has not become the dominant research strategy in information systems research (Cheon et al., 1993; Teng and Galetta, 1990; Vogel and Nunamaker, 1990). Two reasons for this state of affairs are the lack of experimental tools and doubt about the validity of experimental results. Doubt of validity of experimental research especially concerns the widespread use of students as players (Hughes and Gibson, 1991). However, the fact that students react different from managers, or that results from field studies differ from experimental results (Dennis et al., 1990) does not invalidate the results of experimental research. It is adequate if success of an information system method in an experimental setting is positively correlated to success in the field. In medical research, new treatments that are harmful to rats are not dispensed to humans, although they might be quite harmless in view of the difference between rats and man.

Education in information systems methods consists mainly of exercises in an artificial, rather than natural setting, such as case analysis, specification writing and programming, but exercises where the system has to be designed as well as used in a contrived environment are rare. It is obvious that an environment that is sufficiently realistic for experimental research is also adequate for education purposes. Educational and research concerns also coincide because research results can be directly derived from information systems courses (Davis, 1990).

The remainder of this paper is structured as follows: In section 2, Infogame, a game specially designed for research and education in information systems development, is described. Section 3 describes the use of Infogame in a number of courses at Tilburg University. In section 4, conclusions are drawn and directions for further research are given.

2 INFOGAME

2.1 Conventional management games

Though management games have recurrently been used in experiments in information systems, scant consideration has been given to the design of management games specifically for that purpose. In many instances, existing management games have been used, especially for experimental evaluation of decision support systems (Courtney et al., 1983; Dickson et al., 1977; Sharda et al., 1988; Van Schaik, 1988; Yeo and Nah, 1992). A principal characteristic of management games is that players compete in a market. However, this poses problems in evaluation, because results are dependent on actions of competitors. Accordingly, some researches prefer to use dummy players (Van Schaik, 1988) or noncompetitive games (Van der Meer and Roodink, 1991).

Management games have originally been designed to teach subjects such as marketing, finance and accounting. Accordingly, information processing in a management game can be seen as an integral part of decision making. For example, a player who wants to increase his market share may have to decide whether to decrease his price, improve quality, increase advertising or add sales outlets. To this end, the game administrator supplies data such as price, quality, advertising outlays and number of sales outlets of competitors. Further processing of those data by sorting, computation of percentages or indices or graph painting, which in the real world is accomplished by managerial assistants with the aid of spreadsheets, does not fundamentally change those outputs. The essential data processing operations have been effected beforehand according to inherent rules of the game.

The design of a management game where information processing by the player is essential starts from the notion that managers use *aggregated data*. In conventional management games, such data are provided by a simulation model that produces aggregate outputs from aggregate inputs. For example, quarterly production may be defined by a function of planned production, machine capacity, number of employees, and materials purchasing budget. Details like production stoppages as a result of shortage of materials cannot be represented in this model, although they may significantly influence overall profit. In my view, a good information system should signal such problems, and consequently, the conventional management game offers inadequate opportunities to distinguish good and bad information systems designs.

2.2 Outline of Infogame

The design of Infogame was started in September 1985. A preliminary description was given in (Casimir, 1986). The first version, with the users manual and a full description was completed in the beginning of 1989. From that moment, corrections and improvements have been inspired by users. For example, the labour market model has been changed when the original model was found inadequate, and the approach to sales orders that could not be delivered in time was changed twice. From the start, Infogame has been designed as a model that directly simulates individual transactions. This makes data processing necessary, and it also engenders more realism.

The main entity in Infogame is the firm, which is managed by a player or player team. A firm produces and markets one or more products. To produce a product, a technology is chosen from a set of technologies supplied by the game administrator. Each technology is defined by the machine and the number of employees employed, the materials used, the production capacity and the product quality. A product is either an end product marketed in one or more noncompeting industries, or a material used in producing another product. Production and purchase of materials are controlled by setting a *reorder level* and an *order quantity*. Whenever the stock of a finished product or a material is below the order level, a production or purchasing order for the amount specified by the order quantity is executed. To market a product, price and advertising budget are defined. Sales are made to simulated consumers who choose among competitors on a number of determinants, such as price, quality and marketing effort. Other player decisions are investment in machines, demand for loans, minimum and maximum number of employees (the difference permits firms to follow the European habit of diminishing personnel without resorting to forced redundancies), credit terms for customers, and choice of suppliers. All decisions are made for a quarter. A full list of decisions is give in table 1.

Table 1. Instructions

Instruction	Contents
Define output signals	Select signals
Invest in machines	Select machine type(s) and number of each type
Scrap machines	Select machine(s) to be scrapped
Design a product	Select, change or add technology for product(s)
Remove a product	Select product to be removed
Define product characteristics	Set price, reorder level, order quantity, advertising budget
Set parameters for materials	Set reorder level, order quantity
Choose suppliers	Choose supplier(s) for each material
Define terms for employees	Set salary, minimum and maximum number of employees
Ask loans	Ask for loan or credit limit from bank
Repay loans	Select loan(s) to be repaid prematurely
Accept credit and set credit terms	Ask for supplier credit and decide on consumer credit

During a quarter, a firm may produce some tens of product batches and effect some hundreds of separate sales transactions for each product. Every single event may be reported separately in the report file. The player decides what types of events are reported. For example, a firm with cash sales only has no need to separately record sales and sale payments. Apart from collecting event data, a player can collect inventory data at specified intervals. This allows the construction of simple information systems that monitor stocks and utilisation rates. Because in reality, data cannot be collected free, the game administrator can attach a cost to the collection of a data item. Because Infogame uses discrete event simulation, the number of events is finite, so there is a finite number data items that can be collected.

All aggregated data, such as quarterly sales, production or profit must be computed by a separate information system that is supplied by the game administrator or designed by the player. Apart from this information system, Infogame contains an operational information system that, for example, computes the cash level to determine interest charges and computes the stock level of materials and finished products to determine whether sales are allowed, production should be started or purchase orders must be placed. Data in this information system are not directly accessible to the player. For example, the number of unfilled sales orders is not available as a data item.

2.3 Game versions

In the full version of Infogame, all firms start from scratch and must choose in which industries and technologies they want to invest. Industries are characterised by the possibility of stocks and/or back-orders, the number of customers, the average size of orders, etc. By choosing appropriate data, the game administrator can simulate real-life industries. For example, service industries are characterized by the absence of stock, small order size, and a low ceiling on acceptable delivery times. For first-year students, the full version was considered too difficult. Accordingly, the game was played in a restricted version with a single industry with production for stock only, without production of materials. Moreover, in this version all data are recorded without cost. To facilitate testing the information system, the first four quarters were simulated with standard data.

3 EDUCATIONAL EXPERIMENTS

3.1 Infogame courses

Tilburg University offers, among others, four-year curricula leading to degrees (roughly equivalent to U.S. masters degrees) in *business economics* and *information systems*. The business economics curriculum (BE) focuses on accounting, marketing, finance and organization, and contains an introductory course in information systems. The information systems curriculum (IS) contains similar courses, with more emphasis on information systems, and, in addition, a number of courses in Computer Science subjects. A special place in both the BE and IS curriculum is awarded to the subject *quality of information systems*, which deals with accounting information systems and introductory auditing. In all courses, Infogame was introduced primarily for educational purposes, but at the same time, use of the results for exploratory research was planned from the start. The courses where Infogame was used are detailed in Table 2. All courses are described in section 3.2 through 3.5.

Table 2. Use of infogame

Course	Deg-Year	Semesters	Task	See
Decision Support Systems	IS-4	Fall 90-93	DSS Design	3.2
Introduction to information systems	IS-1	Spring 93,94	IS Design	3.3
Quality of information systems	IS-2,BE-2	Spring 93	Specs writing	3.4
Introduction to information systems	BE-2	Fall 93	Parameter selection	3.5

3.2 DSS Design

DSS is an elective course for information systems students in their final year. It has been taken by 8 to 12 students per course. The main purpose of the course is to teach student to design, implement and use DSS. Students are free to choose a programming language or application package, but spreadsheets are used earlier in the course for simpler problems. Students work either alone or in groups of two. The full game is used, and students start by analyzing the data provided by the game administrator for investment selection. The number of players has been too small for statistically significant conclusions, but results suggest that players often underrate the amount of data needed for effective decision support, and that huge losses are often caused by simple errors, such as forgetting to order necessary materials, and not to erroneous strategic decisions. This, in turn, stresses the importance of information systems that can discover such errors.

3.3 Information Systems Design

In the first-year course "Introduction to information systems", Infogame was introduced to give students some practical experience with the system development life cycle (Hice et al., 1974; Sommerville, 1992). Each of the twenty groups of three or four students had to design and use a complete information system for a simple version of Infogame with a single industry producing for stock only. The first four quarters were played according to a fixed scenario and results were made available to students from the beginning of the exercise. Students had to follow a linear development method, and intermediate reports were discussed with student-assistants. A spreadsheet was used for implementation because this was the only package which was known to all participants. When the systems were completed, four additional quarters were played, and after that, students had to write a final report. From an educational point of view, application of Infogame has been a success because it increased students' awareness of information systems design problems. Preliminary results also suggest a positive correlation between the quality of products of consecutive stages, which gives support to the validity of the waterfall model of system development.

3.4 Specifications writing

Infogame was introduced into the course "Quality of information systems" to teach students to make specifications for information systems and to evaluate information systems. The relevant research question is to determine what factors cause user satisfaction with information systems. 71 groups of 2 to 4 students specified an information system for a simple version of Infogame with one industry producing for stock with the possibility of back-orders. Next, an information system purporting to embody those specifications was written by the game administrator. The students used this system to analyze the results of the first four quarters. Decisions for the fifth quarter were based on this analysis. After the game program for the fifth quarter was executed, students evaluated the results and rated the information system. A problem in this procedure was that a large number of programs had to be written, often from inconsistent or incomplete specifications. Because it was decided beforehand that the specifications should not be reviewed, slight flaws in specifications, such as an omission of the frequency of reports, could have dire consequences. The grades assigned to the quality of information systems sometimes represented a view on the game instead of a view on the simulated environment. For example, low grades for reliability were given because the network used did not adequately protect data. The output of 52 groups could be used, and no significant correlation with either profit or length of program was found. Because the number of students in this course is large enough to provide statistically significant results, it will be modified for improved research results. Instead of tailor-made programs, the standard program described in 3.5 will be used, instead of one round, four rounds will be played, and better instructions for quality evaluation will be given.

3.5 Parameter selection

In the second-year course "Introduction to information systems", playing Infogame was optional. 20 groups of 3 or 4 students participated in this exercise and played the simple version of Infogame mentioned in section 3.4. The aim was to show the relation between specification and implementation of information systems. First, each group had to define the specifications of an information system. Next, a standard package was provided, and each group had to select up to ten different reports, each of which contained data on one to ten variables. Because of the small number of participants and some problems with installing the game and the standard problem, no statistically reliable results could be found. However, the experiment was a success from an educational point of view, and it gave some suggestions that preliminary specifications may disagree with features deemed important after implementation. For example, all groups used the financial accounting module, though not every group included accounting reports in the specifications.

4 CONCLUSIONS AND FURTHER RESEARCH

After some initial problems, Infogame provides a stimulating and realistic environment for student work in information systems design. Consequently it can be used as an instrument for research on student reactions to information systems development variables such as the relative advantages of prototyping versus linear development, differences between end-user programming and use of standard programs, or the benefits of CASE tools. Generalization of those results calls for experiments with other groups of subjects, such as managers and information systems practitioners, as well as field studies. By using Infogame to identify promising areas for field studies, the efficiency of field research may be increased. Finally, interesting results may be derived from the use of Infogame in different cultural settings. To this end, the game is offered free to educational institutions.

References

Casimir, R.J., "DSS, information systems, and management games", *Information and Management*, Vol 11. No 3 (Oct 1987), pp. 123-129.

Cheon, M.J., V. Grover and R. Sabherwal, "The evolution of empirical research in IS, a study in IS maturity", *Information and management* Vol 24 (1993), pp. 107-119.

Courtney, J.F., DeSanctis, G. and Kasper, G.M., "Continuity in MIS/DSS laboratory research: the case for a common gaming simulator", *Decision Sciences*, Vol. 14 No. 3 (July 1983), pp. 419-439.

Davis, J.S., "Integrating student learning and faculty research through use of students as subjects", *Journal of Research on Computing in Education*, Spring 1990, pp. 369-375.

Dennis, A.R., Nunamaker, J.F. and D.R. Vogel, "A comparison of laboratory and field research in the study of electronic meeting systems", *Journal of MIS*, Vol 7 No 3 (Winter 1990-91), pp 107-135.

Dickson, G.W., Senn, J.A., and N.L. Chervany, "Research in management information systems, the Minnesota experiments", *Management Science*, Vol 23 No 9 (May 1977), pp. 913-923.

Eppen,G.D., Gould,F.J. and C.P. Schmidt, "Introductory Management Science", 2nd ed. Prentice-Hall, Englewood Cliffs, N.J., 1987.

Gries, D., "Teaching calculation and discrimination: a more effective curriculum", *Communications of the ACM*, Vol 34 No 3 (March 1991), pp. 44-55.

Hamilton, S. and B. Ives, "MIS research strategies", *Information and Management*, Vol 5 (1982), pp. 339-347.

Hice,G.F., Turner, W.S. and L.F. Sashwell, "System Development Methodology", North Holland, Amsterdam, 1974.

Hughes, C.T. and M.L. Gibson, "Students as surrogates for managers in a decision-making environment: an experimental study", *Journal of MIS*, Vol 8 No 2 (Fall 1991), pp. 153-166.

Lapin, L., "Quantitative Methods for Business Decisions", 4th ed., Harcourt Brace Jovanovich, San Diego, 1988.

Sharda, R., Barr, S.H., and J.C. McDonnel, "Decision support system effectiveness: a review and empirical test", *Management Science*, Vol 34 No 2 (Feb 1988), pp. 139-159.

Shneiderman,B., R. Mayer, D. McKay and P. Heller, "Experimental investigations of the utility of detailed flowcharts in programming", *Communications of the ACM*, Vol 20 No 6 (June 1977), pp. 373-381.

Shogun, A.W., "Management Science", Prentice-Hall, Englewood Cliffs,N.J., 1988.

Sommerville, I, "Software Engineering", 4th ed., Addison Wesley, Wokingham, 1992.

Teng, J.T.C. and D.F. Galetta, "MIS research directions: a survey of researchers views", *Data Base*, Vol 20 No 1 (Fall 1990), pp. 1-10.

Van der Meer, F.B. and T. Roodink, "The dynamics of automation: a structural constructionist approach", *Informatization and the Public Sector*, Vol 1 (1991), pp. 121-141.

Van Horn, R.L., "Empirical studies of management information systems", *Data Base*, Vol 5 (1973), pp. 172-180.

Van Schaik, F.D.J., "Effectiveness of Decision Support Systems", Delft University Press, 1988.

Vogel, D.R. and J.F. Nunamaker, "Group decision support system impact: multi-methodological exploration", *Information and Management*, Vol 18 (1990), pp. 15-28.

Vogel, D.R. and J.C. Wetherbe, "MIS research: a profile of leading journals and universities", *Data Base*, Vol 16 No 1 Fall 1984), pp. 3-14.

Yeo, G.K. and F.H. Nah, "A participants' DSS for a management game with a DSS generator", *Simulation and Games*, Vol 23 No 5 (Sept 1992), pp. 341-353.

SA++: How to bring organizational aspects into teaching analysis and design

Jens Kaasbøll

Department of Informatics, University of Oslo
P.O.Box 1080 Blindern, N – 0316 Oslo, Norway
jens.kaasboll@ifi.uio.no

Abstract

Because of the close connection between computer systems and organizations, organizational aspects should be taught in system development courses. Dataflow diagramming is a frequently taught technique that can model formal aspects of information processing. Socio-technical methods are intended to cover relations between people and technology in an organization. To avoid learning two techniques, the dataflow technique has been extended with two organizational aspects from socio-technical methods: goals and exceptions.

Two versions of the extended technique have been used in a university course for systems analysis and design. Organizational issues were taught together with the technique. The students' work has been evaluated.

The technique is sufficiently easy to learn for students. The students also learnt to involve users in analysing their tasks through a wall graph session. However, many students did not learn sufficiently well to consider organizational impacts of computer systems or to design systems to fit organizational requirements. A possible way to cope is to give the students better background in knowledge of organizations.

1 Introduction

It is known that attention should be given to organizational aspects during development of information systems, and that user organizations should be involved in the development. Students should therefore learn to consider both organizational and technical aspects when they learn to analyse and design information systems.

Textbooks on systems analysis and design emphasize that computer systems should meet organizational needs. The techniques for development taught in courses often focus on technical aspects. For example, dataflow diagrams and data modelling are two widely taught techniques that focus on data processes and data.

Methods bringing together technical and organizational issues have been created (eg. Mumford, 1983; Avison and Wood-Harper, 1990). However, it takes time to learn a larger method, and there is not always time for this in academic or industrial courses.

Structured Analysis (SA) is widely used in courses. The method is also frequently employed in system development, although studies show that developers freely adapt the method and its techniques and break its rules (Dekleva, 1992; Bansler and Bødker, 1993). The spreading of

CASE tools that support the method constitutes a reason for believing that SA will continue to be used both in development and education for a period of time.

Given that students will continue to learn to analyse and design information systems in SA courses, there is a need for a way to make the students consider organizational aspects while analysing and designing. In addition, the students should also learn a way to involve users in the development. This paper proposes and evaluates a way to integrate organizational analysis into courses in which dataflow diagramming from SA has a central role.

2 Course Requirements

Textbooks in SA (eg, Delskov and Lange, 1990; Yourdon, 1989; Hawryszkiewycz, 1988) focus on analysis and design with dataflow diagramming as its central technique. In the original version (DeMarco, 1978), the diagrams were intended to describe four versions of an information system: current physical system, current logical, new logical, and new physical. Recent versions have emphasized the analysis of the new, logical (essential) structure of the information system.

The logical dataflow diagrams capture the formal flow, processing, and storing of data. The physical diagrams may be used for portraying the formats and means for flow and storing. In addition, the physical processes can represent the persons and the computers that perform the logical transformations. The flows and stores do not describe all data structures. This can be done by means of data modelling or object oriented modelling.

The requirements for a course where students learn to handle organizational and technical issues are:

1. During work with the dataflow diagrams, the students have to consider organizational impacts of current systems.
2. When shaping a new system, the students should learn to make the system fit organizational requirements.
3. There should be a way for students to learn how to engage user organizations in the development process.

3 Setting of the Experiments

An extension of dataflow diagrams called SA++ was developed to be a remedy for meeting the requirements. The extension is motivated in section 4 and described in section 5.

The first evaluation of SA++ was carried out in 1992 in a university course on system analysis and design. This experiment is described and evaluated in section 6.

This course was intended to meet the requirements nos. 1 and 2 above. In addition, the test aimed at answering the following question:

0. Can SA++ be comprehended and used within an acceptable time limit?

The second experiment was carried out during the same course in 1993. This time the course intended to meet requirements nos. 2 and 3. Description and evaluation are found in section 7.

4 Organizational Aspects

There are a multitude of organizational aspects that could be considered in analysis and design. The choice of aspects to be included in the dataflow analysis is based on the shortcomings of the dataflow diagramming and the opportunities for making improvements that can be learned easily and used effectively.

4.1 Exceptions

The processes in dataflow diagrams describe the routine transformations from input to output. Practitioners often say that they have minor problems with the routines; the design problems are created by all the exceptions. This is confirmed by research, which also emphasizes exceptions. Gasser (1986) has analysed the fit between work and computing in detail in two organizations. He has identified how people work around formal routines. Socio-technical studies also point at the need for finding the discrepancies or variances from routines (Mumford, 1981, p.12). The processes will therefore be extended with exceptions from the formal transformations.

Gasser has identified three areas of accommodation to misfit. These will be used to characterize misfit between routines and the actual work.

Fitting. Fitting work consists of making changes to computer arrangements or adjusting the routines of work. Fitting work constitutes a deviation compared to the routines. But after making the fits, the routines might have been changed.

Augmenting. Augmenting work is to undertake additional work to compensate for the misfit.

Working around is when using the computer in ways for which it was not designed, or when avoiding routines or computers by performing ad hoc. (Gasser 1986, pp.214–216)

Carrying out a process with exceptions presumes that the process is performed by a person, not by a machine.

4.2 Goals and Results

The physical processes describe who or what performs a process and the logical processes describe the formal transformations. The question about why a process is carried out can be answered by referring to the rules of the process for the routine processing. There has to be another way of describing what is determining the exceptional processing. When behaviour in organizations cannot be explained by reference to a rule, the actors can be assumed to have intentions, goals, preferences, values, etc., as their reasons for action.

Whether or not an actor is behaving in a certain way because of a goal is not very relevant for the analysis and design of an information system. However, it is relevant to know whether a specific process contributes to fulfilling the goals of the organization, departments, stakeholder groups, users, customers, clients, or other constellations of people relevant to the organization.

Socio-technical methods emphasize the goals of people in organizations (Mumford, 1981, Mumford, 1983), focusing on efficiency and job satisfaction. Effectiveness and customer satisfaction are other categories of goals which may be linked to strategic business objectives.

Even if an effort has been made to make organizations rational by means of computer systems, decisions are made on grounds that may be all but rational and uniform (Feldman and March, 1981). The difficulties of understanding the complex pattern of goals may be used as an argument against considering goals at all in analysis and design. However, if strategic goals, competing interests, and personal motives are kept in the shadow, there is a risk that a designed system will be ignored.

When considering the goals of a process, there is an opportunity to compare whether the results of the process fulfils its goals, and if not, there may be a deviation that could be explained by an exception. To make the comparison, it is also necessary to know the results of the process.

For the formal transformation of a logical data process, the result is as specified in the output. Also, when regarding the persons or computers in the physical data flow, other results may be noted, amongst them unintended effects. An example is given below.

4.3 Other Organizational Issues

There are many other factors missing in the diagrams. It might, eg, have been useful to consider social relations between people performing the processes, the resources available for keeping the various parts of the information system going, or the organizational culture. However, to keep the diagrams as simple as possible, exceptions and goals have been selected as their only extensions. An example will illustrate that some other organizational factors, including skills and resources, can be considered by the extensions proposed.

5 SA++

Since the diagrams are extended with two issues, exceptions from regular performance, and goals and results of the actual performance, the extension is called SA++. The extension has been presented in lecture notes for a university course (Kaasbøll, 1992). It will be illustrated by a case.

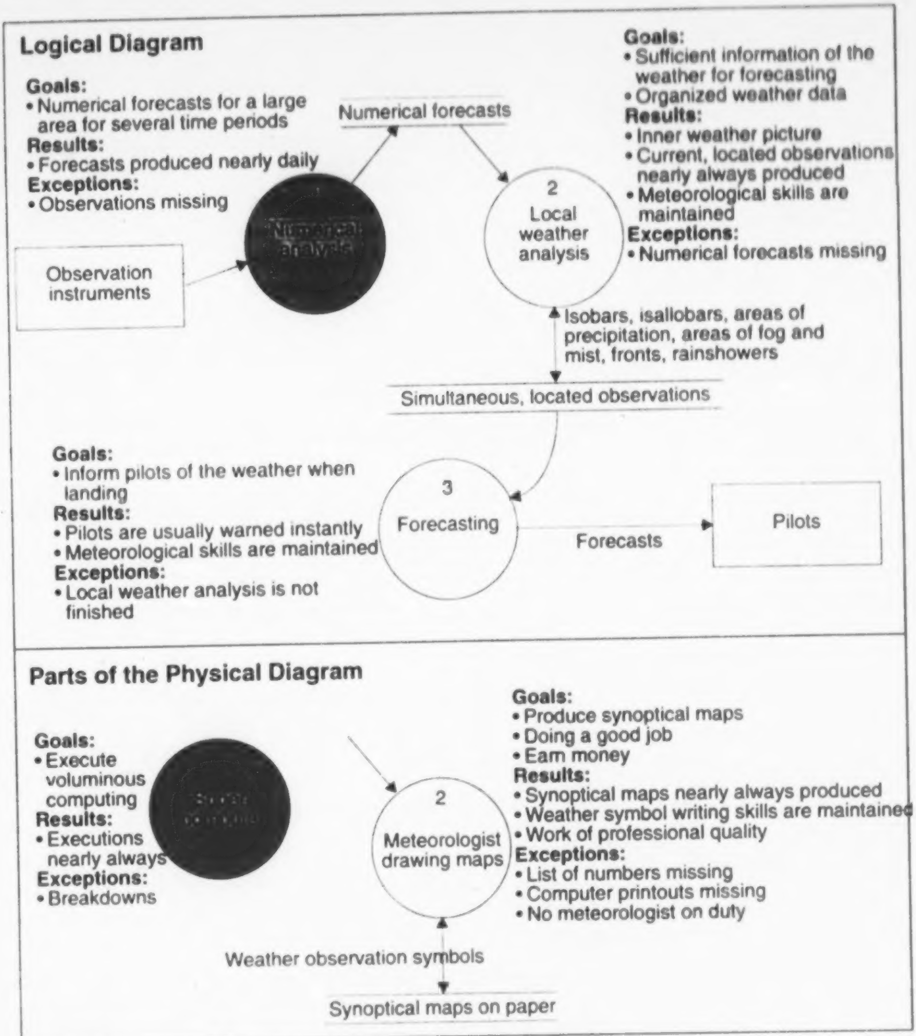
> The meteorologist at an airfield analyses the weather by reading observations from various sources, including numerical forecasts of a larger region produced at a central meteorological institute. The meteorologist draws a synoptical map showing the weather situation. He simultaneously forms an inner weather picture, which is his conception of the weather development in all three dimensions of the atmosphere over a period of time which stretches into the future. The meteorologist uses his inner weather picture for making forecasts, issue warnings, and briefing pilots. He needs a comprehensive understanding of the weather in order to do this. (Perby, 1987)

Figure 1 shows parts of an extended dataflow diagram for the case. Logical diagrams should model the formalizable data flow. The diagram therefore models the contents of the synoptical map that the meteorologist draws, Simultaneous, located observations. At the physical layer, it is appropriate to call it Synoptical map, since this tells how the data are arranged. Since the "inner weather picture" is the information which the meteorologist creates from the data, it neither appears in the processes nor in the flows.

Goal covers any goal of the process that may exist in the organization and any intention of the persons undertaking the process. Result describes the output, products and other changes produced by the process. The results may satisfy goals, they may be insufficient, or they may not be related to the goals at all, the latter being typical for side-effects.

Process no.1, Numerical analysis, is performed by a computer, as indicated in the physical diagram and the gray shading. Its goal is related to the needs of the meteorologist in this case. Others may have different goals for the same processing.

Process 2, Local weather analysis, is performed by the meteorologist. The two first mentioned goals and results of this process refer to the information and the data. The third result, Meteorological skills are maintained, is a longer term result that is emphasized by Perby in her study of the meteorologists (Perby, 1987, p.219–221). Similar side-effects are described in process 2 in the physical diagram. These results are described as side-effects of the processes, because Perby gives no hints that the meteorologists consider skill maintenance while doing


Logical Diagram
Goals:
• Numerical forecasts for a large area for several time periods
Results:
• Forecasts produced nearly daily
Exceptions:
• Observations missing
Numerical forecasts
Numerical analysis
Observation instruments
2
Local weather analysis
Goals:
• Sufficient information of the weather for forecasting
• Organized weather data
Results:
• Inner weather picture
• Current, located observations nearly always produced
• Meteorological skills are maintained
Exceptions:
• Numerical forecasts missing
Isobars, isallobars, areas of precipitation, areas of fog and mist, fronts, rainshowers
Simultaneous, located observations
Goals:
• Inform pilots of the weather when landing
Results:
• Pilots are usually warned instantly
• Meteorological skills are maintained
Exceptions:
• Local weather analysis is not finished
3
Forecasting
Forecasts
Pilots
Parts of the Physical Diagram
Goals:
• Execute voluminous computing
Results:
• Executions nearly always
Exceptions:
• Breakdowns
2
Meteorologist drawing maps
Goals:
• Produce synoptical maps
• Doing a good job
• Earn money
Results:
• Synoptical maps nearly always produced
• Weather symbol writing skills are maintained
• Work of professional quality
Exceptions:
• List of numbers missing
• Computer printouts missing
• No meteorologist on duty
Weather observation symbols
Synoptical maps on paper

▇ Automated parts. Shading suggested by Sutcliffe and McDermott (1991)

Figure 1 Extended logical dataflow diagram

their daily analyses and forecasts.

The goals of the processes performed by persons, partly determine the transformations going on in the processes. When no exceptions occur and rules for performing the process exist, the dataflow diagrams express that the process is performed according to the rule indicated by the process name. For processes for which no rule exist, or when exceptions occur, the goals should determine the outcome of the processes.

According to the rules of dataflow diagrams, the persons, organizational units or machines that perform the processes are described as physical processes. Therefore, goals relating to the work of persons or organizational units in general are described in the physical layer. A goal like Doing a good job may be closely related to the forecasts given. That indicates that this goal could have been placed in the logical diagrams or in any process performed by the meteorologist. It is placed in the physical diagram, because it then becomes clear to whom this goal belongs. If an aggregated diagram had been made, in which the meteorologist just appeared in one process, such goals should be placed there.

5.1 Deviations

The goals, results and exceptions are compared during the next step of the extension: deviations and analysis of dependencies.

The goal deviations have already been indicated: insufficient results and side-effects. Current, located observations nearly always produced is an example of an insufficient result, while Meteorological skills are maintained is a side-effect.

The analysis should proceed in the following way:

1. Compare goals and results, and note goal deviations of the processes. Compare information about goals and results compiled from different sources. If the goals differ, is it because

   (a) the users come from different parts of the organization, and their goals correspond to those of their organizational unit?

   (b) the users have their own interests or represent groups of interest with goals that differ from those of the organization, or are in conflict with those of the organization?

   (c) the users have no explicit goals of their work?

2. The processes should be classified either as necessary or augmenting work or working around the computer system (Gasser, 1986). Augmenting work and working around should be compared with the goals. What seems superfluous in one setting, may be necessary for other goals.

3. Determine dependencies between the factors found. In particular, do routine deviations cause goal deviations? (Kaasbøll, 1992)

Four simple chains of dependencies for the example are shown below:

Deviation from routine, augmenting: Extra effort is needed in local weather analysis because numerical forecasts are missing due to breakdown in the super-computer.

Deviation from goal: Current, located observations are not produced in local weather analysis because there is no meteorologist on duty.

Deviation from goal: The meteorologists build inner weather pictures and maintain their skills.

Deviation from goal: The pilots are not warned instantly because local weather analysis is not finished.

5.2 Reducing the Deviations

System development may have several aims. The extended dataflow analysis may be used in design to reduce deviations. Two designs are proposed to illustrate the consequences for results, exceptions, and deviations.

The first proposal aims at reducing the latter deviation from goals. The delay of the weather forecast is explained to happen because the local weather analysis has to be completed before forecasts are given. The local weather analysis lasts for one hour when performed by the meteorologist. A design option is therefore to substitute the meteorologist with a computer program that make numerical analysis and forecasts that are adequate for pilots. To avoid technological changes in aeroplanes, the meteorologist will still have to transmit the forecasts to the pilots on orally. Parts of the design proposal is sketched in figure 2.

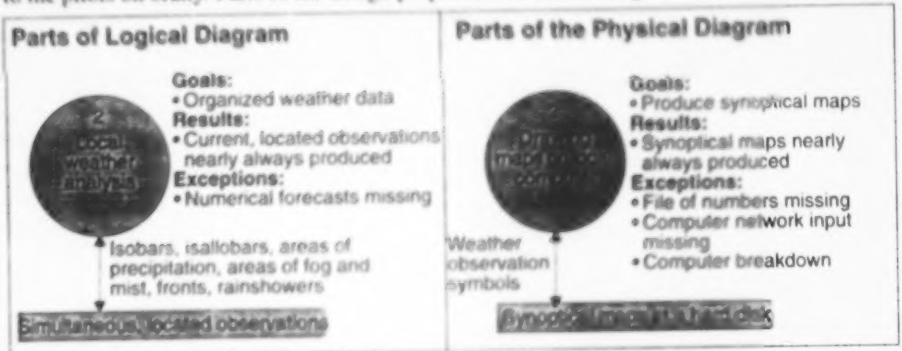

Parts of Logical Diagram
2
Local weather analysis
Goals:
• Organized weather data
Results:
• Current, located observations nearly always produced
Exceptions:
• Numerical forecasts missing
Isobars, isallobars, areas of precipitation, areas of fog and mist, fronts, rainshowers
Simultaneous, located observations
Parts of the Physical Diagram
Goals:
• Produce synoptical maps
Results:
• Synoptical maps nearly always produced
Exceptions:
• File of numbers missing
• Computer network input missing
• Computer breakdown
Weather observation symbols

Figure 2 Redesigned system. 1. alternative. The local weather analysis is performed by a computer.

The new design has not altered the routine deviation of the old system. The goal deviation caused by No meteorologist on duty has only been slightly altered, because the new system is also dependent on meteorologists informing the pilots. The side-effect being that the meteorologists build inner weather pictures is removed, and their skill maintenance is reduced. Perby and the meteorologists warn against this solution, because they believe the meteorologists will loose their skills in the long run (Perby, 1987, pp.223–224).

Another design aims at reducing the latter goal deviation whilst keeping the skill maintenance. The proposed implementation is to replace the paper for drawing maps with a computer screen, and let the data transfer between process no.1 and no.2 be electronic. The deviations are as follows:

Deviation from routine, augmenting: Extra effort is needed in local weather analysis because numerical forecasts are missing because of breakdown in the super-computer.

Deviation from goal: Current, located observations are not produced in local weather analysis because there is no meteorologist on duty.

Deviation from goal: Meteorologists build inner weather pictures and maintain their skills.

5.3 Scope of Organizational Issues

The SA++ technique covered skills and contents of work in this example. The timeliness of the output is also considered, in addition to computer breakdowns. Job satisfaction in general can be covered, since persons are included. The efficiency of processes and the effectiveness of the results can also be considered. In other examples than the one chosen, the exceptions may be more dominating.

6 First Course Evaluation

The first evaluation of SA++ aimed at answering the following questions, corresponding to the requirements in section 2.

0. Can SA++ be comprehended and used within an acceptable time limit?

1. Did students learn to consider organizational impacts of current systems?
2. When shaping a new system, did the students learn to make the system fit organizational requirements?

The evaluation was carried out in 1992 in a university course on system analysis and design; 69 students participated. The course was scheduled to take half of the students' time during the spring term. The course material contained guidelines with explanations of how to extend dataflow diagrams with goals and exceptions, and of how to find goals and exceptions in organizations. The activities involving extended dataflow diagrams were class room exercises, description of an actual information system, and parts of a written exam. A class room exercise in SA++ was to make extended dataflow descriptions of the weather forecasting case, based on the text by Perby (1987). Three weeks were spent on introduction to SA and to the extensions.

The students knew that SA++ was a novelty. However, they were not informed that their learning process was part of an experiment.

The guidelines for finding goals encompassed a brief introduction to types of goals. Many of these issues were known from previous courses. The method for finding routines, goals, results and exceptions in organizations was divided into four steps, inspired by Hawryszkiewycz's search procedure (1988). It can be summarized as follows.

1. Read documents about the organization, for example, annual reports, organizational charts, documents describing goals, commercials.
2. Interview with managers. Ask the managers about goals for their organizational unit, and let them give an overview of the routines.
3. Observation and interview with those working with the information. Let them *show* you what they do in their actual work at their normal workplace. Ask what is going on, for instance, how do these data arrive to you, how do you know how to do it, how many copies do you make and who are they intended for? After a survey of their routine, ask about exceptions: eg, what if something is missing, what are the usual problems, what happens if you forget to sign, makes it extra work for you if someone is absent?
4. Observe a meeting concerning their objects of work. To understand what happens at the meeting without interrupting, make this your last technique of investigation. Prepare by studying the material for the meeting beforehand.

Groups of approximately ten students found organizations on their own, which they described with SA++. They were tutored by a graduate student instructor.

Each group produced a twenty page description of an information system in an organization. The descriptions were swapped between the student groups, and the groups were given the task to design a computer system for an organization which another group had described.

Method

A qualitative analysis of the knowledge that the students documented in their project reports was carried out (Larsen, forthcoming). Eight out of sixteen student designs were arbitrarily chosen to make the analysis a manageable task.

The method of analysis was to disclose the knowledge upon which the students had based their design decisions. 75 design decisions were found in two textbooks (Avison and Wood-Harper, 1990; Yourdon, 1989). It was analysed whether the students had made these decisions, and if so, on what ground they had made them.

560

Results

The students had made most of the decisions concerning functionality proposed in the textbooks. These decisions were supported by the diagrams.

Four of the eight groups had not analysed the goals sufficiently for making design decisions. Decisions concerning social aspects and socio-technical aspects were made in lesser extents than proposed in the textbooks. Some of these decisions were made without firm grounds, while others were grounded in the goals and exceptions in SA++. Other decisions, eg. concerning implementation, security, and maintenance were mostly neglected by the students.

Discussion

The four groups that had analysed goals properly had learnt SA++ within the time available during the projects. In many instances, the students mixed physical and logical concepts, however.

The students were given the task to make an extended dataflow description based on a text during the exam. The average grade at this question did not differ significantly from the average score for the total questions at the exam. The exam showed that extended dataflow diagramming technique can be learned and used during three weeks of teaching and some days training before an exam. Question no. 0 is therefore confirmed.

Only half of the groups had learnt to analyse organizational impacts sufficiently to answer confirmative to question no. 1. The remaining students were not capable of or spent too little effort in finding organizational aspects.

The same half of the students who learnt to find goals, also managed to propose an innovative design that would fit organizational requirements, according to Larsen (forthcoming).

The fact that the students neglected several types of decisions that were proposed in the method, is a consequence of the lack of attention concerning these issues in the course and in SA++.

7 Second Course Experiment

The technique was redesigned to improve creativity and user participation (Wiig, 1994), thus intending to fulfil requirements nos. 0, 2, and 3. Freely drawn wall graphs were used in initial descriptions. Exceptions were noted when appropriate, and goals were structured on separate descriptions. Dataflow diagrams were made later on.

The revised version was used in the course the following year. This time, the systems were described in sessions where both users and students participated.

The students responded mixed experience as to finding goals in a questionnaire. Eg, it was difficult because the organization had no clear goals, it was enlightening to discuss goals, it made little sense in a small organization. The actual users who participated in the wall graph sessions answered in a questionnaire that they experienced being engaged in the wallgraphing.

The designs seemed more innovative than the year before. A preliminary analysis shows that the proposed systems were designed to meet some of the organizational goals and to reduce exceptions.

The results of the exam did not differ from the previous course.

The conclusion of the second experiment is that requirements nos.0 and 3 were met, while no. 2 only was achieved by some of the students.

8 Conclusion

A technique has been developed to learn students in SA courses to consider organizational issues during analysis and design.

The technique is found to be sufficiently easy to learn for students. The students also learnt to involve the users in analysing their tasks through a wall graph session.

However, many students did not learn sufficiently well to consider organizational impacts of the computer systems or to design systems to fit organizational requirements. A possible way to cope is to give the students better background in knowledge of organizations.

Bringing organizational issues into dataflow diagramming is an example of modification of a technical oriented system development technique. If, eg, object-oriented methods are taught, organizational aspects could probably be included in a corresponding way.

Acknowledgements

Many thanks to Bodil Bye Larsen and Ingrid Wiig, who have provided invaluable help in the development and evaluation of the technique. Thanks also to Kristin Braa, who has supported the experiment through unlimited encouragement, and to Sara Selmark for improving the language.

References

Avison, D.E. and Wood-Harper, A.T. (1990) "Multiview: An Exploration in Information Systems Development" Blackwell, Oxford

Bansler, J. P. and Bødker, K. (1993) A Reappraisal of Structured Analysis: Design in an Organizational Context" *ACM Transactions on Inf. Sys.* 11, 2, pp.165–193

Dekleva, Sasa M. (1992) The Influence of the Information Systems Development Approach on Maintenance *MIS Quarterly* Sept.1992, pp.355–372

Delskov, Lis and Lange, Therese (1990) "Struktureret Analyse — Integreret Systemanalyse" Teknisk Forlag, København

DeMarco, Tom (1978) "Structured Analysis and System Specification" Yourdon, New York

Feldman, Martha S. and James G. March (1981) Information in Organizations as Signal and Symbol *Administrative Science Quarterly* Vol. 26, pp.171–186

Gasser, Les (1986) The Integration of Computing and Routine Work *ACM Transactions on Office Information Systems* Vol.4, No.3, pp.205–225

Hawryszkiewycz, I.T. (1988) "Introduction to Systems Analysis and Design" Prentice-Hall, NY

Kaasbøll, Jens (1992) "SA++" Lecture Notes, Department of Informatics, University of Oslo

Larsen, Bodil Bye (forthcoming) "Supporting Design Decisions by Means of SA++" Master Thesis, Department of Informatics, University of Oslo

Mumford, E. (1981) Participative Systems Design: Structure and Method *Systems, Objectives, Solutions* 1, pp.5–19

Mumford, E. (1983) "Designing Human Systems for New Technology. The ETHICS Method" Manchester Business School, Manchester

Perby, M.-L. (1987) Computerization and the Skill in Local Weather Forecasting *in* "Computers and Democracy: A Scandinavian Challenge", Bjerknes, Ehn, and Kyng (eds.), Avebury, Aldershot, pp.213–229

Wiig, Ingrid (1994) "Methodological Support for User Participation and Creativity in System Development" Master Thesis, Department of Informatics, University of Oslo

Yourdon, Edward (1989) "Modern Structured Analysis" Yourdon Press, Englewood Cliffs, NJ

BEST COPY AVAILABLE

A CASE Environment for the Teaching of Information Systems Principles

Gary Allen and Adrian R Jackson

School of Computing and Mathematics, The University of Huddersfield, Queensgate, Huddersfield, HD1 3DH, England

Abstract

In this paper we describe the CASE Environment which is used to support a staged strategy for the teaching of Information Systems Principles at the University of Huddersfield. It is no longer possible to teach Systems Analysis and Design without the use of computer based support, but the range of commercially available tools are, for a variety of reasons, unsuitable for our purposes. Hence we have developed a set of tools giving graded and staged support to our educational strategy. These tools are based on proprietary meta-CASE technology and function as stand-alone tools supporting single techniques (such as Entity-Relationship Diagrammers), loosely coupled toolsets supporting methods (such as Shlaer and Mellor's OOA) and tightly integrated toolsets supporting the same methods but applying extensive semantic constraints and integrity checking. The result of teaching using these tools is that the students achieve not only an understanding of the methods which are used and the interrelations of their components, but also an understanding of the manner in which tools are used to support the development process. In this paper we describe the environment and tools that have been developed, together with their relationship to the strategy on which they are based. Our intentions for future refinement, research and development of the environment are also outlined.

1. Introduction

The software development process has obtained such complexity over the relatively short period of time since rigorous system development procedures have been available, that the only practical way to manage that complexity is by the application of computers. One outcome of this application is the increasing use of Computer Aided Software Engineering (CASE) tools and environments. For example a method such as SSADM [NCC91] is now of such size and complexity as to be beyond application by a single or small group of developers, to anything other than the simplest projects.

This leaves educators with a problem: we can no longer teach Information Systems Development without the use of CASE tools and CASE environments, but using such tools introduces another topic to the syllabus, we must teach about these tools as well as with them. The tools currently available support single methods such as SSADM, Information Engineering [Martin81], HOOD [Booch86] etc. This presupposes that we wish to teach methods from this range (In fact we will always want to teach novel techniques that have not yet achieved market acceptance). Two further problems are the high cost of such software, particularly when multiple versions are needed for different methods, and the fact that the tools are geared up to full scale industrial use, not to educational use. The drawbacks of proprietary CASE systems and the requirements for educational CASE technology have been elucidated previously [Allen92a, Allen92b, Jackson92]

We need to teach students :

- Principles of systems development and methods,
- Instances of systems development methods,
- Principles of CASE technology,
- Use of CASE tools,
- New methods for which there is no CASE support.

Meeting these requirements with proprietary tools is impossible due to the extreme cost involved (purchasing tools for each method), the nature of the tools (expected full scale use) and in the last case, unavailability. The solution we have adopted has involved the use of meta-CASE technology. We have determined our requirements for CASE in education. This has led us to a phased and staged approach to the teaching of IS principles. The teaching of these principles is supported by tools which we have developed using CASE tool generators. This approach allows us to develop (after a considerable initial expenditure) new tools to support individual components of an IS development method rapidly. In addition we can easily integrate these tools to produce complete development environments and apply extensive semantic constraints and consistency checking to the systems which the students develop. A final bonus is that we have achieved a high degree of modularity within these methods which allows a degree of interchangeability of development techniques. For example we have a range of tools for describing an entity's behavior over its lifetime, e.g. SSADM style Entity Life Histories and Finite State Automata. The appropriate tools for the task may be plugged in and out of the environment for a particular method, allowing us to teach the technique we find most appropriate.

In the following section we describe the meta-CASE technology that we have chosen, the educational strategy we have adopted, the current state of our tool set and our plans for change and progress

2. Meta CASE

Over the past 3 - 5 years the focus of academic research has shifted from the development of method specific IPSE and CASE tools to the area of Meta-CASE : i.e. the development of tools which in turn facilitate the production of customised CASE tools and support environments.

Meta-CASE tools provide an environment for the development of customised CASE tools. Typically, Meta-CASE environments consist of two main components : a (CASE) tool which enables the abstract definition of the requirements of the target software, and a run time environment capable of turning that abstract specification into an executable implementation.

Central to the Meta-CASE philosophy is the provision of generic diagram- and structured text-editors. A generic diagram editor is a parameterisable tool which may be instantiated for any diagrammatic notation. Such tools typically treat any notation as a directed graph of nodes and arcs. By

564

providing facilities for the definition of the shapes of the nodes, the shapes of the arcs (links) and simple connectivity rules, editors may be derived for virtually any diagrammatic notation.

Similarly, a generic structured text editor will typically provide a 'layout language' for the definition of the structure of text. The editors thus defined may be very simple, for example defining a screen where the end user of the generated tool fills in details for a data dictionary such as entity name, reference number and attributes, or as complex as, for example, a structured text editor for a high level programming language such as Ada, where the tool allows the user to edit the source code only in terms of the language's underlying syntax.

Given the above generic editors, Meta-CASE tools must also provide a navigation mechanism to allow the definition of permissible navigations between the different diagrammatic editors and text panes which make up the end user CASE tool. Most Meta-CASE products provide navigation facilities via a hyper-text style of interaction. Navigations may be defined between diagrammatic and text panes to enable the user of the generated tool to navigate around the tool as required.

Finally, some mechanism must be available for the tool builder to incorporate method-specific functionality into the generated tool. A generic diagram editor, for example, may be used to define the syntax of a particular notation and, to a limited extent, the semantics of the technique. However, a mechanism is required which enables the tool builder to extend the generic facilities where required to allow comprehensive method-specific syntactic and semantic checking to take place.

An example of Meta-CASE technology, in use at the University of Huddersfield, is the Ipsys Toolbuilder. The Ipsys product range is essentially the commercial outcome of the Alvey funded ECLIPSE project [Bott89]. Toolbuilder is the latest product in the Ipsys Meta-CASE catalogue and builds upon the functionality provided by the Tool Builders Kit (TBK). TBK itself provides facilities for the development of stand alone tools and integrated toolsets for the development of IPSEs. It incorporates a proprietary database based upon the entity-relationship-attribute model with inheritance; an applications interface based upon the X Motif look and feel; a generic diagram editor and generic structured text editor parameterised by the use of three proprietary languages; a data manipulation language; and a message handling system. Furthermore, as the TBK is implemented using published C libraries, there is the option to use C as an integral part of any developed tool.

TBK can be used for the development of stand alone tools or tools integrated in the IPSE sense via a common database and common user interface. However, for the development of comprehensive method specific CASE tools, the Toolbuilder software provides far more functionality. Toolbuilder has itself been developed using the TBK and is a true Meta-CASE product offering the following features [Ipsys92] :

- A (CASE) tool, METHS, for the capture of method specifications which is used to maintain a 'methods database'.

- A 'publisher' tool used to describe the structure and format of reports to be produced by the generated tool; the descriptions are held in a template database.

- A set of generator programs known collectively as the 'deasel generators', which generate tool data and compile the generated tools from the methods database.

- A 'deasel library' which is linked with the generated tool data to produce executable on-line and off-line tools.

3. The Educational Strategy Adopted

The tool support strategy adopted at the University of Huddersfield involves the provision of three stages of tool support :

1) Generally available, stand-alone tools, used to support all of the techniques and notations taught to the students throughout their course of study. Such tools represent little more than an automated pen and paper, forcing the students themselves to consider issues such as the syntactic and semantic accuracy of the diagrams produced. At the same time, these tools enable students to become familiar with the look-and-feel, or user-interface style, of the chosen support software without the additional level of complexity inherent in a full support environment.

2) Intermediate level support in the form of a CASE tool designed to support the particular notations used by the techniques of a given method, but which do not provide automated support for issues such the semantic constraints of individual diagrams, consistency between diagrams, and completeness of a proposed solution to a problem. The low level of method-specific support offered by these tools is intended to force the students to think about the techniques used and about the cross-referencing and consistency checking which must take place.

3) A fully integrated (CASE) support tool providing the sort of method-specific syntactic and semantic checking found in most commercial products, but developed specifically for use in education and thus free from the problems associated with the use of commercial tools in education [Allen92a, Allen92b, Jackson92]. Such tools allow the students to concentrate upon the problem in hand, and upon the development of a feasible solution to that problem, rather than upon the application of the method itself or the internal consistency of the solution proposed. Such tools should have an identical look and feel to their intermediate-level counterparts, the organisation and layout of the menus should be the same, but the features offered should be extended to incorporate full method support.

The rationale behind the development of this three-stage tool support strategy was as follows : It is the norm to teach software development by firstly introducing students to a number of wide-spread, commonly used techniques such as data-flow modelling and entity-relationship diagramming (or data-modelling). The provision of simple, stand-alone tools at this stage is intended to give the students an opportunity to familiarise themselves with the hardware, operating system and support software used throughout the course of study, while at the same time developing skills in the application of these common techniques.

Once the fundamentals have been covered, the next stage in the educational process typically involves the formalisation of the software development process by the introduction of a coherent software development method. The constituent techniques and notations of the method are introduced to the students one-by-one for individual study. Most of these techniques will already have been covered, and so this stage is concerned largely with the idiosyncrasies of the techniques as they relate to the method as a whole. The intermediate level tools are used to support these individual techniques and notations as they are introduced. Typically, exercises will be used to allow the students to practice the techniques and thus to begin to understand them. Such tools allow each individual technique to be addressed in isolation, thus removing the additional complexity inherent in the use of many commercial products, which are unable to support such selective use of the facilities provided.

Once the majority of the constituent techniques and notations have been covered, it is usual then to introduce students to the cross-referencing and consistency checking required by the method. The use of the intermediate tools at this stage requires the student to do the thinking - no short cuts are yet provided. These tools can be used to support short case-studies which require the application of the entire method, thus allowing the students to begin to address the problems of the internal consistency of the solution proposed.

Once the method has been covered in full, large case-studies may be used to reinforce the concepts introduced. By this stage, the students should be fully aware of the constituent techniques of the method and of the consistency checking required. The focus at this stage is upon the application of the method to industry-scale problems. A full CASE tool here allows a concentration upon the application of the method itself whilst at the same time introducing students to the sort of facilities provided by CASE.

4. Current Tool Provision

At present, stage one tools are provided to support the majority of techniques and notations used during the first and second semesters of all courses offered by the School of Computing and Mathematics. These include Data Flow Diagrams, Entity-Relationship Diagrams, State Transition Diagrams, Jackson style Structure Charts, and SSADM style Entity-Life Histories. These tools are openly available to the students to use as and when required.

At the level 2, intermediate tool level, we currently offer support only for a modified version of Shlaer and Mellor's Object Oriented Analysis Method [Shlaer88 & 92] (OOA - for a discussion of the reasons behind the adoption of this method as a vehicle for teaching object oriented analysis, see [Jackson92]). It is our intention to develop further tools at this level to support alternative methods including Yourdon's Structured Analysis and Design [Yourdon89a & 89b].

Finally, at the level 3, fully integrated CASE level, we have an OOA toolset developed in-house specifically for use in teaching, and we also have commercial tools available to support the SSADM and HOOD methods. These later tools have been developed using the Toolbuilder software and so present a similar user interface to those developed in-house, but are not considered ideal for use in education as they suffer from many of the problems previously identified with the use of commercial products in this field. Again, it is intended to extend the range of tools offered at this level, with support for the Yourdon method being the current priority.

In addition, we also use the TBK and Toolbuilder software both to support final year undergraduate projects in CASE development and evaluation, and to support research activities within the School, including on-going work in the area of methods integration.

5. Future Plans

Our plans for the development of the environment involve the extension of the environment to support the remaining phases of the software lifecycle and extension to support alternative development methods. In the short term, this involves the development of intermediate level tools and full CASE toolsets to support methods such as Yourdon and JSD [Jackson83] and the development of support for processes later in the lifecycle. Our main problem in the short term is the support of the design process, before we can address this we need a satisfactory model of the design process which is applicable to a wide range of design methods. Also the system will need the introduction of mechanisms for version control and configuration management other than simple sequential versions as currently implemented. Only after these problems have been addressed are we ready to look at problems involving programming, testing and verification.

Research into the development of CASE tools to support methods integration (i.e. the integration of structured and formal methods into a unified framework [Allen et al 92]) is currently underway in collaboration with Leeds Metropolitan University.

6. Conclusions

We have found that in order to use CASE successfully in our teaching it is essential that some education about the nature of CASE tools and technology must be provided, this obviously increases the size of our courses. CASE tools are also prohibitively expensive for use in Higher Education Institutions. Our solution to this problem has fitted nicely to our other problem of finding suitable Information Systems development environments and tools for use on undergraduate and postgraduate degree programmes in Computing. Meta-CASE software has enabled us to develop environments and tools catering specifically for our needs at costs significantly below those of purchasing tools, (which would also, inevitably, be a poor fit to our requirements). We can develop new tools in a matter of hours and support for Systems Analysis methods with full semantic constraint and consistency checking can be provided in 5 weeks for a typical method such as Shlaer and Mellor's OOA.

References

[Allen et al 92] Allen, P M, Jackson, A R, Rawson, M L, Evans, A S & Semmens, L T, *Formalising Object Oriented Analysis - Modelling the World in Mathematics*, Internal Research Report, The Polytechnic of Huddersfield 1992

[Allen92a] Allen, G, *Specification of the Requirements for Pedagogical Software Development Methods*, Internal Research Report, University of Huddersfield, 1992

[Allen92b] Allen, G & Jackson, A R, *Unifying Principles of Software Development - A Method for use in Software Engineering Education*, Proceedings Information Systems Teaching - Improving the Practice (ISTIP92), Sunningdale, September 1992

[Booch86] Booch, G, *Object Oriented Development* ,IEEE Transactions on Software Engineering 12(2) pp 211-221

[Bott89] Bott, F, (Editor) *ECLIPSE - An Integrated Project Support Environment* , IEE, 1989

[Ipsys92] *Ipsys Toolbuilder Methods and Concepts Manual*, (c) Ipsys Software plc, Malborough Court, Pickford Street, Macclesfield, Cheshire

[Jackson83] Jackson M A, *System Development*, Prentice Hall, 1983

[Jackson92] Jackson, A R & Allen, G. *A Pedagogical Information Systems Development Method*, Proceedings 3rd International Conference on Information Systems Developers Workbench, Sopot, Poland, 1992 pp 367-375

[Martin91] Martin, J & Finkelstein, C. *Information engineering*. Savant Research Studies, 1981

[NCC91] *SSADM Version 4 Reference Manuals*. NCC Publications 1990

[Shlaer88] Shlaer, S & Mellor, S J. *Object Oriented Analysis Modelling the World in Data*. Yourdon Press 1988

[Shlaer92] Shlaer, S & Mellor, S J. *Object Lifecycles Modelling the World in States*. Yourdon Press 1992

[Yourdon89a] Yourdon, E. Modern *Structured Analysis*.Prentice Hall International 1989

[Yourdon89b] Yourdon, E & Constantine, L. *Structured Design* (2nd Edition) Prentice Hall International 1989

THE CURRICULAR CONTENT OF INTERNET TRAINING PROGRAMS IN THE UNITED STATES OF AMERICA

Lester J. Pourciau

The University of Memphis, University Libraries, Memphis, TN 38152, U S A

Abstract

The rapid growth and development of the Internet, along with an ever increasing array of services available over the Internet, has created the need for various programs of training for potential Internet users. Evidence abounds that such training programs are widespread. Some few have been conducted or distributed over the Internet, but the vast majority are conducted within individual facilities where the Internet is used. These include colleges and universities throughout a large portion of the world, and also many commercial firms which make use of Internet capability. This paper identifies the frequency of mention of various elements of Internet user training as these appear in a variety of sources.

1. INTRODUCTION

When students and faculty return to the campus of The University of Memphis for the fall semester, 1994, they will find a new library building fitted with cutting-edge, state-of-the-art information handling and processing technology. This will be the culmination of planning which began several years ago and resulted in a design to provide no fewer than 517 individually wired workstation locations throughout the new building. Terminals and workstations will initially be installed at 210 of these locations. A vast array of capability will exist for those persons making use of this equipment.

On the most basic level, users will have access to a friendly online public access catalog (OPAC). Not only will this access be to the OPAC of library holdings at the University of Memphis, but also will allow access to the OPACs at the local public library, the Christian Brothers University Library, the library of the State Technical Institute at Memphis, the Shelby State Community College Library, and that of Rhodes College, all of which are located in the city of Memphis. These other libraries all use the same automation system planned for the UMem Library and, because of this, users will use the same screens and search protocols regardless of which libraries' OPACs they search. Interinstitutional agreements, already extant between UMem and these other institutions, will greatly facilitate the availability of a wide array of material to students and faculty in the Memphis area.

Moving beyond the OPACs of local area libraries, students and faculty using the new library at UMem will additionally have access, through terminals in the library, to the academic mainframe in Computer Services, and thus have access to the Internet. The growth and development of the Internet races along at an almost dizzying speed and, at the time of this writing, it allows one to search hundreds of library catalogs throughout the United States and the world. Searching these from the terminals in the new library building will be a simple matter, and will be restricted only by the individual user's familiarity with languages other than English. Just as the library at UMem will allow access to library catalogs all over the world, so will users in other libraries have access, over the Internet, to the UMem OPAC.

As dramatic as might be the foregoing, it is nonetheless a strong signal to make serious note of that kind of preparation necessary to allow students and faculty to make use of the wide array of technological capability to be installed in this new library building. Learning the basic use of the automation system to be installed in this new building will not be a particularly difficult matter. Responding to the variety of questions and frustrations that will be experienced by library users as a result of their efforts to interact with the Internet is quite another matter. A facetious comment on such frustration was made earlier this year by Mike Royko, a syndicated newspaper columnist whose satire appears in newspapers all over the United States and Canada:

> Then there is something called the Internet, which is a worldwide hookup of thousands of computer networks. The Internet is already an information superhighway, except that you have to be a full-fledged computer nerd to navigate it. I have been there. It's like driving a car through a blizzard without windshield wipers or lights, and all of the road signs are written upside down and backwards. And if you stop and ask people for help, they stutter in Albanian.[1]

Royko's comment is humorous but illustrates well the frustrations experienced by the uninitiated when attempting to use various services and databases available via the Internet. Not many individuals stop to reflect on the fact that a database complete with a variety of search protocols and procedures developed in Hong Kong bears no necessary or coordinated relationship with a different database created and mounted on some computer in Ottawa, Canada. To be able to locate these different databases, requires a certain level of expertise and to interact with them, to manipulate them, or to download them requires yet an additional increment of expertise.

2. METHODOLOGY

The origins of this paper first began to materialize when reflecting upon the impact of information handling processing technology on American higher education and, particularly, as it will impact on faculty and students in the next several months at The University of Memphis. With the foregoing in mind, an effort was made to identify Internet training programs in use on college and university campuses, in commercial organizations, and in special libraries throughout the United states. The basic approach to identifying such programs consisted of posting on the Internet through various discussion lists or listservs, a request for examples for individual training programs as these were in use in various locations. A variety of responses were received to this request. Some of the responses were in electronic format and received via e-mail, and others were mailed via surface mail and on paper. What follows is a discussion of the various responses received, and a comparison of the elements common to all of the programs identified, or elements which appeared in the majority of the programs identified.

Preliminary to a discussion of the responses received, it is appropriate to characterize briefly the use of electronic media in attempting to conduct any form of survey investigation. In the particular circumstances described in this paper, a request was posted on seven different discussion lists or listservs. This request indicated that the writer was attempting to determine programs of Internet User Instruction in various educational institutions, commercial organizations, or governmental offices. It asked specifically for copies of course syllabi or course outlines.

It is difficult to claim statistical validity or validity of representation when conducting a survey such as was done for the purposes of this study. To be sure, one can determine the number of individual subscribers to any listserv or discussion list but, at any point in time, any number of those subscribers may have no role of any kind anywhere in any Internet instructional activities. The only listserv known to this writer which speaks specifically to the interests of those involved in Internet training is Nettrain.[2] A variety of respondents to the requests posed by this writer came from individuals who reviewed the requests on the Nettrain Discussion List.

3. RESULTS

In all, there were forty-three different responses received answering the posted requests. While this number may appear to be small and certainly cannot, at least in any statistical sense, be argued as representative, it, nevertheless, allowed for a sufficient number of examples of syllabi or outlines so that some reasonably confident conclusions might be reached about what is judged to be the optimum array of components in an Internet User Training Program in the United States.

Of the forty-three responses received, eight were requests that the writer share the results of this survey. Of the remaining thirty-five responses, twenty-six provided a syllabi, an outline, or some discussion of the content of a course or training program in use. Nine of the responses suggested various training material available at various Gopher sites. There was some duplication in these nine recommendations.

Of the various responses received, several which included information about instructional offerings were barely a page in length and, in the most general fashion, only mentioned one or two topics. The most extensive of the responses mentioned each of the elements listed in Table 1, which identifies topics and their frequency of mention in the various responses received. Indeed, the most extensive of the responses included, not only discussion of each of the elements listed in Table 1, but also a variety of other more esoteric topics, and even included material germane to the principles and fundamentals of computer science. For the purposes of the discussion here, Table 1 identifies only topics which were mentioned at least five times in the collective responses received. Approximately fifteen additional topics were mentioned only once or twice, and many of these are, as was stated above, more appropriately subjects in the field of computer science.

Table 1.

Training Topic	Frequency of Mention
Telnet and Hytelnet	31
Gopher	25
FTP	24
Introduction to the Internet	21
E-mail	21
Archie	15
Veronica	14
WAIS	11
Discussion Groups (Listservs)	11
World Wide Web	9
Usenet News Groups	7
VMS	6
Bitnet	5

4. DISCUSSION

There are no major surprises in Table 1, at least in terms of the individual topics mentioned five times or more in the various responses included in this survey. Yet, one might have expected to see electronic mail mentioned more than twenty-one times in thirty-six usable identifications of instructional programs, and it is of some small curiosity that Veronica and Gopher did not receive the same frequency of mention; neither did Archie and FTP. The basic reason for the discrepancy in mention frequency associating Gopher and Veronica, and Archie and FTP, is that several of the syllabi, outlines, or discussions which mentioned Gopher and/or FTP were sufficiently brief so as not to include Veronica and Archie as tools for searching Gopher space or files mounted on FTP servers.

The list of items mentioned fewer than five times and not represented in Table 1 includes signature files, Netfind/White pages, Jughead, DOS, UNIX, clients and servers, ASCII format, Domain Name System, Gateways, IP Address, NIC (Network Information Center), NII (National Information Infrastructure), NREN, TCP/IP, TN3270, Z39.50 Protocol, and several word processing software packages, Wordperfect being the more commonly mentioned.

The results reported above are, at first glance, disappointing. They are disappointing because one might have expected more responses from a request which was posted to seven different discussion groups.[3] When reflecting upon the question of why there were only forty-three responses to a request posted to literally thousands of subscribers to seven different listservs, some interesting thoughts occur. The fact that several of the responses were no more than one page in length, that several of them were requests to share the results of this brief survey, and that a few of the responses were significantly thorough in detail and substantial in terms of the subject matter included, all indicate that Internet training is an activity which is relatively new to virtually all persons who attempt to undertake it. Another way of expressing the newness of Internet training is to compare it, for example, with the study and teaching of the literary works of Wolfgang Goethe. Those scholars who pursue the work of Goethe write, criticize, and build upon a substantial body of literature which has evolved and matured over a long period of time. Such is not the case with the Internet which really has not been around very long, and has not been the subject of discussion, evolution, learning, and training as has been the subject matter of many other fields. In spite of the sophistication of expertise on the part of many people such as those who, for example, participate in postings to NETTRAIN, the evidence internal to the various responses received by this surveyor suggests that training programs, in spite of areas of considerable substance, are essentially in their infancy.

A few numbers from Win Treese's "The Internet Index" illustrate very well that the Internet is growing enormously and is maturing very rapidly.[4] The December, 1993 revision of the Internet's Index indicates an annual growth rate for Gopher traffic on the "Net to be 997 percent; the annual rate of growth for World-Wide Web traffic to be 341,634 percent. In 1993, the number of mail messages carried by IBM's Internet Gateways was 340,000. The number of Internet hosts in July, 1993 was 1,776,000. These numbers are interesting, particularly those which express rates of growth. Growth occurs typically in areas where there is room for greater maturity, greater sophistication, and greater substance. At the risk of sounding cynical, one might say that the Internet, along with all of the activities which take place on it, are in a state of postpubescence at this time. There is nothing wrong with this because, by all accounts, the Internet will increasingly be of value and assistance to the global village.

The need for user training is expressed frequently and widely by many persons. Nancy Szofran has spoken lucidly to the need for training offered by libraries and librarians. She commented particularly on a model course of instruction used by the University of Montana.[5] In a lengthy and thoughtful review of The Whole Internet User's Guide & Catalog, Christine Borgman comments that "while there are societies and coordinating boards, no one entity has final authority over 'the Net,' making it quite difficult to learn what's out there. Nothing approaching an 'Internet Catalog' yet exists where one can search for people and information resources. As the resources proliferate so do the tools to locate them--Gopher, Archie, Veronica, Jughead, Telnet, Hytelnet, Wide-Area Information Servers (WAIS), World Wide Web, Campus Wide Information Servers (CWIS), lists of listservs, lists of online catalogs, etcetera." She closes her review by stating that "we are far from the day when we have a single, friendly interface that will provide a full-featured window to the Internet for all users. Until that day comes, we need ambitious books such as Krol's, despite their flaws."[6]

At the 1993 National Online Meeting in New York, debate included approaches to teaching students the intricacies of online searching.[7] Further, the cover story appearing in the February/March, 1994 issue of the Bulletin of the American Society for Information Science is "Navigating the Networks: The Training Hurdle."[8]

5. RECOMMENDATIONS AND CONCLUSION

It can easily be argued that the frequency of mention of any topic in a discussion is indicative of the importance assigned to that topic and of the interest in it. In light of this, the obvious recommendation to be made on the basis of this brief, pilot study is to include those items mentioned most frequently in proposed programs of internet training. Certainly, the first several listed in Frequency of Mention, Telnet, Gopher, and FTP, are included in virtually all programs of an introductory nature. Others on the list are found in programs of an intermediate or advanced nature.

The investigation reported here took place approximately one year before the actual dates of ISD'94. Many people, when speaking about the internet, suggest that anything said on any given day about it, is outdated the following day. While this would not be entirely true for the results reported here, some increasing growth and development of the internet has indeed taken place since the original survey. Of particular note is the emergence of the World Wide Web and Mosaic. Indeed, a recent call for papers for a special issue of Journal of Intelligent Systems: Integrating Artificial Intelligence and Database Technologies on Network Information Discovery and Retrieval remarks that "The Recent Explosive Growth of Information and The Internet, fueled largely by the popularity of the World Wide Web and Mosaic, has opened up new possibilities and requirements for network access to information.[9] Also a recent issue of EDUPAGE included "Internet Tidbits." Among these tidbits is the statement that "There are sixteen hundred copies of Mosaic downloaded from NCSA each day."[10] With Mosaic as a very popular client software used increasingly in a large number of locations, it is important that it be included in intermediate and advanced internet training curricula.

This paper reports on an attempt to conduct a preliminary or pilot survey of Internet User Training Programs in the United States of America. As reported in the preceding discussion, the results of the survey were, in one sense, disappointing but, in other senses, indicative of the state of maturation of Internet user training as an activity. It is certain that we can anticipate rapid development, sophistication, and maturity of this activity as time goes by.

5. REFERENCES

1. Mike Royko, February 16, 1994.

2. NETTRAIN@UBVM.CC.BUFFALO.EDU

3. ASIS-L@UVMVM.BITNET
ELEASAI@ARIZVM1.BITNET
JESSE@ARIZVM1.BITNET

LABMGR@UKCC.BITNET
LIBADMIN@UMAB.BITNET
NETTRAIN@UBVM.CC.BUFFALO.EDU
NETADMIN@LEO.VSLA.EDU

4. Win Treese, "The Internet Index," 12/16/93 (TREESE@CRL.DEC.COM)

5. Nancy Szofran, "Internet Etiquette and Ethics," Internet Librarian, January 1994, pp. 68-69. The training outline she mentions is available from FTPHOST.UMT.EDU/INTERNET/TRAINING.

6. Christine Borgman, review of Ed Krol's The Whole Internet: User's Guide & Catalog, Sebastopol, CA: O'Reilly & Associates; 1992, in Journal of the American Society for Information Science, 45(2):117-119, 1994.

7. Paul Blake, "News, The Internet and Student Training: Debate at the National Online Meeting," National Online Meeting, New York, NY, 1993, in Online & CDROM Review, 1993, Vol. 17, No. 3.

8. "Navigating the Networks: The Training Hurdle," Bulletin of the American Society for Information Science, February/March, 1994, pp. 15-24.

9. Marianne Winslett, "CFP: Special Issue on Network Information Discovery and Retrieval," ASIS-L Digest, 5 Aug., 1994 to 8 Aug., 1994.

10. EDUPAGE, 08/07/94, (IN%"info@ivory.educom.edu")

A VSM BASED PLAN FOR FIRST YEAR INFORMATION SYSTEMS COURSE AT THE UNIVERSITY OF PAISLEY

Angus John Quin

Department of Computing and Information Systems, University of Paisley, Paisley, Renfrewshire PA1 2BE, United Kingdom

Abstract

Higher education in Scotland and the UK as a whole is going through a period of enormous change. Expected changes in the patterns of enrolment include 50% increase by the end of the decade with female and mature students contributing mainly to this growth (MacFarlane 1992) (The Association of Graduate Recruiters). Many courses will have to become more flexible enabling them to be matched to meet mature student's experience and needs (MacFarlane 1992) (Fairclough 1993). Technology based learning and teaching methods such as multimedia, simulation and distance learning as well as resource sharing have roles to play. One question is how can the creation and maintenance of high quality be reconciled with the increasing effectiveness currently required (MacFarlane 1992). The Viable Systems model is used to analyse the structure of two different courses with the aim of improving the educational experience of the students with diverse backgrounds whilst coping with increasing numbers.

1. INTRODUCTION

The current recession is forcing graduates to seek jobs at every level of industry. This diversification of graduate skills is expected to benefit the economy (The Association of Graduate Recruiters). Re-structuring of industry has resulted in many middle management jobs disappearing as many organizations evolve a flatter management structure. This means graduates must abandon the concept of a planned career and must be prepared to accept more responsibility for their own career development (Fairclough 1993). The emphasis today is on the ever increasing need for specialists who can work in teams who combine entrepreneurial, business management, product improvement, client relations and project management skills (Fairclough 1993). Higher Education should develop a more appropriate approach by encouraging graduates to acquire attributes such as initiative, independent thinking, organisational and time management skills, problem identification and solving skills and presentation skills (The Association of Graduate Recruiters) (Binks 1994).

There is an increasing awareness that the United Kingdom's prosperity, vitality and international standing depend on its becoming a more highly educated nation (Alway 1990). This has resulted in a number of reports advocating closer collaboration between higher education, industry and professional groups to achieve those goals (The Association of Graduate Recruiters) (Fairclough 1993) (Alway 1990) (Wright 1990) (Nicholson 1989). Subjects such as mathematics, science and technology are seen by many as being central to this success (Alway 1990) (Nicholson 1989). The implications for this realisation are that a more diversified higher eduction system is needed. It should provide learning opportunities for large numbers of students from a much wider segment of the population at every stage of their life (Alway 1990).

There are many possible structures for programmes and courses and there are also many ways of modelling them (Baume and Baume 1992). The MacFarlane report advocates the use of a systems approach in higher education (MacFarlane 1992). This has been the author's approach for some years. Courses in both industry and higher education were designed, developed, organized and managed using concepts from General Systems Theory and in particular cybernetics.

When designing programmes and courses there are all sorts of questions such as:

Q1 Why not let the students run the course?

Q2 Why not let students pick their own projects?

Q3 What is the best way to organise a course giving the student maximum choice?

Q4 Why should getting a degree not be a partnership between staff and students?

Q5 Is there any limit to the size of intake, and if so what is it and what are the limiting factors?

This paper is the result of exploring new ways of managing and organizing courses in an industrial training school and first year classes for a BA in Business Information Technology at the University of Paisley.

2. ONTOLOGICAL BASIS

The assumption from which the author is working is that organizations can be designed in the same way as designing a modern aircraft, car or new computer system. Given that organizations can be designed the next question is what is the best approach? This is a matter of serious debate (Popper 1972) but the general message is to get the people involved as early as possible. In the case of students this approach is a unique opportunity by providing an example of management structures in

578

use which provides a framework for debate and questioning existing management practices. In new computing systems techniques such as prototyping, (Agresti 1986), and client led design have been suggested.

3 THE VIABLE SYSTEMS MODEL (VSM)

Omit this section if you already know Stafford Beer's Viable System Model of an organization.

3.1 Cybernetics in Management

Stafford Beer proposes a new way of looking at the management structure of organizations. Organizations can be a company, consortium, department of education or a national economy (Beer 1991). Stafford Beer started work on the science of organizations, or cybernetics, in the 1950's and has written over 200 items including a number of books (Beer 1979) (Beer 1981) (Beer 1991) (Beer 1984). The VSM model in its present form appeared in (Beer 1979).

3.2 The Basic Model

Beer proposed that the cybernetic model of any viable system should consist of five necessary and sufficient subsystems. These subsystems are necessary if the organization is to be capable of maintaining its identity in an environment shared with other organizations (Beer 1981) (Beer 1984).

A number of people have applied the model in a variety of different situations (Espejo and Harden 1989) (Flood and Jackson 1988). In particular VSM has been used to diagnose complex situations involving technological, social and informational changes (Beer 1991) (Flood and Jackson 1988). This is called Viable System Diagnosis, VSD (Flood and Jackson 1988).

3.2 The Recursive Model

Most organizations are far more complicated than the simple one envisaged above and the basic model has to be extended, see Figure 1.

This system may consist of parts. If it does then each of these parts is autonomous and it must of necessity also be a viable system, (Beer 1979) (Beer 1991) (Espejo and Harden 1989) (Quin 1992) (Williamson and Deasley 1994).

3.3 VSM as a Business Model

The viable systems model can be used as a cybernetic framework to describe and construct business organizations (Beer 1991). Business organizations can be split into strategic units, each unit having its own management and operations. The viable systems model thus provides a environment within which these units can

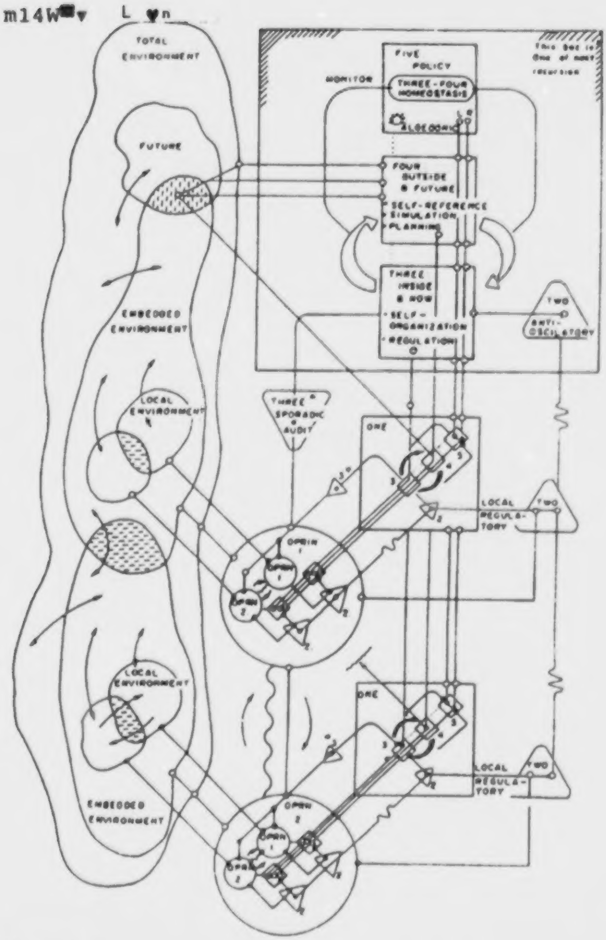

TOTAL ENVIRONMENT
FUTURE
EMBEDDED ENVIRONMENT
LOCAL ENVIRONMENT
LOCAL ENVIRONMENT
EMBEDDED ENVIRONMENT
This box is one of next recursion
FIVE POLICY
MONITOR
THREE-FOUR HOMEOSTASIS
ALGEDONIC
FOUR OUTSIDE & FUTURE
SELF-REFERENCE
SIMULATION
PLANNING
THREE INSIDE & NOW
SELF-ORGANIZATION
REGULATION
TWO ANTI-OSCILLATORY
THREE* SPORADIC AUDIT
ONE
LOCAL REGULATORY
TWO
OPRN 1
OPRN 2
ONE
LOCAL REGULATORY
TWO
OPRN 2
OPRN 1

Figure 1

operate (Beer 1991) (Espejo and Harden 1989) (Williamson and Deasley 1994). The viable systems model is also used to assess the effectiveness of communication by filtering out the noise in messages using the law of requisite variety, (Ashby 1978) (Beer 1979) (Beer 1981) (Beer 1991) (Beer 1984) (Espejo and Harden 1989).

Stafford Beer claims that the viable systems model guarantees stability and provide a dynamic framework which supports change (Beer 1991). It is also claimed to be a diagnostic tool that is self-validating in the sense that engineers and managers can relate to it (Beer 1991) (Espejo and Harden 1989) (Flood and Jackson 1988).

3.5 Criticism

Criticisms of the viable systems model have been summarised as methodological, epistemological and utility (Flood and Jackson 1988). These criticisms claim that VSM is difficult to apply, too goal oriented and is difficult to apply in practice.

3.6 Current developments

Much of the current thinking on VSM is covered in (Beer 1991). Some current applications of VSM are given in (Trans IMC 1992) (Quin 1992) (Williamson and Deasley 1994).

4 VSM AS A PRACTICAL TOOL

This section gives examples of how the viable systems model could be applied to higher education, government, industries and its interaction with the environment.

4.1 VSM and Higher Education

VSM can be used to model existing or planned organizational structures (Beer 1991). For example the proposed bureaucratic structure for higher education in the UK is a recursive model involving the formation of a national body as senior management, with the three higher education funding councils and educational institutions as the next two layers of recursion.

The above model an be extended to model the partnership between educational institutions and industry. Industry is seen as a customer (Alway 1990) (Wright 1990). Marketing, improving product quality, product maintenance and the use of use of automation are now common place in education.

VSM can model a more dynamic management structure by providing a framework for incremental analysis and design of the organization (Beer 1981) (Beer 1991). For example in a typical university there is a management unit headed by a chancellor and board of governors. The operations consist of a number of different faculties, departments and supporting services. The entire structure can be modelled using the viable system model. The VSM framework provides a suitable structure for either reorganising an existing structure or subcontracting out or privatising the service or function it performs.

4.2 VSM as a Tool for Managing a Course

The goal of improving the organization and management of a university level course is summarised by the following objectives:

- increase the involvement of students and staff in the running of the course thus improving both morale and standards

- encourage a more direct staff student dialogue so that problems do not fester but are discussed openly when they arise

- give students better support by freeing staff from lower level work

- encourage the development of leadership skills at all levels

- make better use of valuable resources

4.2 A Course as a Multisystem

In general courses are a complex mixture of organizational, technical and social problems. Courses also have goals which are reassessed from time to time. Control is seen as one of the main problems. Multiple control loops form multisystems involving different layers of recursion. Examples of control loops include tutor/course leader, tutor/group, group/group and group/industry systems. Viable System Diagnosis or VSD is one approach to analysing the system based on VSM (Flood and Jackson 1988).

The notion of recursion is fundamental in both VSM and VSD. Since recursion means that the whole system consists of replicated parts the analysis can start with any unit (Flood and Jackson 1988). In the rest of this paper the management of a single first year course covering the application of information technology in business systems only is discussed.

4.3 Student Groupings

There are many different student groupings in a course. If tutorial groups are used these are restricted to 20 students per group. Within each tutor group the students may form groups of up to three or four members in order to undertake project work. There is no reason why individual students may work alone if they prefer.

If students do decide to study in a group then they automatically create another level of recursion. Each student in the group must negotiate their role with the rest of the group and how they are to be assessed. This helps individuals with disabilities such as cerabal palsy who may have a lot to contribute but do not have all the skills required to successfully complete a project on their own.

4.3 Mode of Study

The VSM negotiating channel encourages students to negotiate their mode of study, and may support distance and self learning. Reasons for this are geographical, family commitments and illness. The mode of study may change during the session due to an accident or other change in student circumstances.

4.4 VSM as a communication Tool

The Viable Systems Model enables the organization of the class to be illustrated showing the student groupings, tutors lecturers and other roles in one diagram. This is usually done in week one of the course.

Whilst remaining faithful to the whole first year each individual class grouping can be analyzed in isolation whilst ensuring basic cybernetic concepts of communication are preserved.

With the growth of IT and its applications we need to remind the students of the multisystem nature of any organization, NOT the view attributed to Computer Scientist's that an organization can be represented as a set of simple systems (Harris 1990). This message can be illustrated by using VSM ideas to organize and control any course.

4.5 Subject Management, Systems 3, 4 and 5

VSM provides students with a framework which enables them to negotiate their role in the management structure This emphasis the "role model" of VSM and demonstrates the metamorphic nature of VSM where the management of the class can pass from lecturer(autocratic) control to class(democratic) control whilst the essential underlying cybernetic structures remain intact. By taking part in these negotiations students use VSM when developing their arguments.

A typical structure could be the subject lecturer in system 5 whilst students man systems 4, 3 and 3*. In practice these roles often change during a session with the lecturer gradually relinquishing control.

4.5 Operations, System 1

Once the management structure of the class has been established the students are then formed into groups. These groups form another layer of recursion. Each group is required to investigate the establishment of a new business or product in an existing organization.

One strength of the Viable Systems Model is that it can be used to model many different organizational structures, for example it could be used to model a traditional hierarchal structure based on a set of sequential operations or functions. Alternatively it can be used to model a product based unit based on using simultaneous engineering approaches (Quin 1992) (Williamson and Deasley 1994).

Finally once the student project groups have been established the student groups can negotiate the order of presentation of material, timescales for course works, resources and tutorial support needed and the standards and frequency of continuous assessments.

This is the strength of the VSM. It makes it possible to discuss with either a single group or the whole class a set of design solutions to the problem of organizing first year and selecting the optimal solution.

The educational benefits help students mature by giving them the responsibility for their own educational environment whilst simultaneously enabling larger classes to be used. Mature students are given an important role to play in as much that their experience is seen to be of use by the class which helps eliminate the problem of "feeling out of place".

4.6 System 2

The lecturer is ultimately responsible for standards but the elected member(s) of the class are responsible for ensuring the class is aware of the standards required. These standards may be imposed by the university. They include rules for using laboratories, smoking, use of the university logo and recommended teaching methods (SHEFC 1993).

In practice the lecturer is responsible for negotiating with the university administration for laboratory and tutorial room resources, though the students will be the motivating force.

Another important role for system 2 is at the term end when everyone wants a print out. By organizing themselves the class can make use of limited hardware resources.

4.7 System 3

Lecturers are usually responsible for monitoring standards during the course but there is no reason why this could not be done by students. One example is if peer group assessment (Woodward 1992) is being used.

Similarly the lecturer and/or students representatives are responsible for identifying student problems. Warnings of imminent problems include projects falling behind schedule.

When disputes occur that cannot be resolved at the group level the reporting mechanism to senior management is invoked and an audit invoked. Disputes may be an altercation between tutors and groups, between groups or members in a group. These disputes would probably require subject leader intervention.

The VSM offers a suitable communications tool for both the students and staff to use to identify and discuss problems.

The above reorganization requires

- a timetable for regular tutorials where objectives, goals and project completion dates are agreed and monitored

- once in a term course committee meetings which provides a forum for student feedback and general course problems are discussed

- a newsletter dealing with common problems and reminding students of important dates and events such as end of term parties, course committee meetings and student union information and who the class representatives are and the role they play

- beginning of term briefing meeting where the students are given the timetables, objectives, guidelines and important dates for the term

5 CONCLUSION

It appears that the VSM a suitable approach to analysing a high volume intake course. Management decisions based on the VSM have been used for the last three years and have worked well, though students are frequently reluctant to assume too much responsibility and some tutors reluctant to relinquish control.

VSM has provided a framework for reassessing how first year courses are organized and managed, VSM ensures better and more democratic use of resources and has proved to be an excellent means of providing students with an explanation of the system they are working in and how to use it and how to model may be used to understand management problems in their selected enterprise.

Further formal research is required before recommendations can be made, however the results to date are encouraging.

References

Alway, B.(1990) *Collaborative Courses in Higher Education, Expanding the Partnership with Industry*, The Council for Industry and Higher Education

Agresti. W.W. (1986) *New Paradigms for Software Development*, IEEE Computer Society, Washington D.C.

Ashby, W.R. (1978) *Design of a Brain*, Chapman and Hall, London

Baume, C. and Baume, D. (1992) *Course Design for Active Learning*, CVCP Universities Staff Development and Training Unit

Beer, S. (1979) *The Heart of Enterprise*, John Wiley, Chichester

Beer, S. (1981) *Brain of the Firm*, John Wiley and Sons Ltd, Chichester

Beer, S. (1984) The Viable System Model: Its Provence, Development, Methodology and Pathology, *Journal of Operational Research*, Vol 35, 1, 7 - 25

Beer, S, (1991) *Diagnosing the Organization*, Wiley, 2nd edition, Chichester

Binks, M. and Exley, K. (1994) *Gaining that extra degree of ability*, Guardian

Espejo, R. and Harden, R. (1989) *The Viable Systems Model: Interpretations and Applications of Stafford Beer's VSM*, John Wiley and Sons, Chichester

Fairclough, Sir, J.(1993) *Review of Engineering Formation*, The Engineering Council, London

Flood, R.L. and Jackson, M.C. (1988) *Cybernetics and Organizational Theory: A Critical Review*, John Wiley & Sons, Chichester

Flood, R.L. and Jackson, M.C. (1991) *Creative Problem Solving*, John Wiley & Sons, Chichester

Harris, J. (1990) *IT Starts, Developing Systems Together*, NCC Blackwell Ltd, Oxford

MacFarlane, A.G.J., (1992) *Teaching And Learning in an Expanding Education System*, Polton House Press, Lasswade

Nicholson, Sir, B. 1989 *Towards a Skills Revolution - a youth Charter* CBI Task Force on Training, July

Popper, K. (1972) *Objective Knowledge*, Oxford University Press, Oxford

Quin, A.J. (1992) Designing Organizations, *Information Systems Workbench*, Sopot

SHEFC, (1993) *Teaching and Learning Technology Programme*, Higher Education Funding Council England, circular 34/93

The Association of Graduate Recruiters, *Roles for Graduates in the Twenty-First Century*, ref N/93/65(a)

Trans IMC, (1992) *Transactions of the Institute of measurement and control*, 14, 1

Williamson, A. and Deasley, P. (1994) Systems thinking and Computer-Integrated Manufacturing, *Systems Practice*, 7, 1

Woodward, J. (1992) *Guidelines for Promoting Effective Learning in Higher Education*, Centre for Research on Learning and Instruction

Wright, P.W.G.(1990) *Industry and Higher Education*, The Society for Research into Higher Education, Open University Press, Buckingham

Special Aspects (Comparisons)

COOPERATIVE RESPONDING TO INDIRECT INFORMATION REQUESTS IN DATABASE RETRIEVAL

Xu Wu[1] and Nick Cercone[1]

[1]Department of Computer Science, University of Regina, Regina, SK S4S 0A2,
CANADA

Abstract

In this paper, we present a knowledge-base framework for generating *cooperative answers* to *indirect queries*. An *indirect query* can be considered as a nonstandard database query in which a user did not explicitly specify the information request. Especially, we present methods for inferring users' intended actions, determining a user's information requests from the user's contextual queries, and for automatically reformulating indirect queries into direct queries. The inference process is carried out on the basis of a user model, called the *user action model*, as well as the query context.

1. INTRODUCTION

The Cooperative Query Answering (CQA) problem has been explored to further the development of *intelligent information systems* (IIS). The difference between the cooperative query answering systems and the traditional database query answering systems is that the former can supply a user informative and useful answers through reasoning about the user's query plan, intention, interesting subjects, presuppositions, misconceptions, and so on (Gaasterland et al., 1992). These reasoning capabilities enable the cooperative query answering systems to respond to user's queries like human conversation (Chu and Chen, 1992; Kaplan, 1982; Kao et al., 1988; Cuppens, 1988).

The objective of our study is to develop a knowledge-based framework for formulating *cooperative answers* to *indirect queries*. An *indirect query* can be considered as a nonstandard database query in which the user did not specify explicitly the information request. In an interactive query answering environment, an indirect query is an efficient means of refining an unsuccessful query plan. Consider a user with a retrieval goal G attempting to realize G through a query plan $P(G)$ consisting of the queries, $Q_1, Q_2, ..., Q_n$. In traditional database query answering systems, if the retrieval result obtained by performing the query plan $P(G)$ is undesirable, e.g., the result is 'null', the user should refine the query plan $P(G)$ and rewrite the queries of $P(G)$ for subsequent retrieval. We call the *specification* for refining the unsuccessful query plan an *indirect query*. Indirect queries can be formulated in the same form as normal queries, but the information request is not specified. In the proposed framework, the system first infers user's retrieval goal from the user's previous queries and automatically rewrites the indirect query, then generates information responses, including *affirmative* and *negative* responses, for the indirect query.

In this paper, the methods for inferring the users' information requirements from contextual queries and for generating informative responses on the basis of these inferences will be described. In order to support the inference of a user's information requirement from the contextual queries, a user model, called *user actions model* has been developed in the system. The user actions model is constructed upon the semantic-network model S-Net (Wu and Ichikawa, 1992) which represents the semantic knowledge about the underlying database and user discourse domain. The user actions model consists of an *action hierarchy*, a set of *Key Concepts tables* and *Key Attributes tables* corresponding respectively to *action nodes* and *case fillers* represented in the *action hierarchy*. The construction of the action hierarchy is similar to that of the *plan hierarchy* (Kautz and Allen, 1986; Vilain, 1990) and *plan library* (Beek and Cohen, 1990). But, the user action model proposed in this paper is different from the plan modeling approach in that each *action node* and *case node* of the user action model is associated with a *Key Concepts table* and a *Key Attribute table* respectively. By incorporating these tables into the plan knowledge-base, the system can efficiently infer a user's goal from the user's queries. Another feature of the user action model is that it is coupled with a semantic-net model in which the semantic information about the database is represented. This enables the system to automatically rewrite user's previous queries.

2. PRIMITIVES OF THE SYSTEM DESIGN

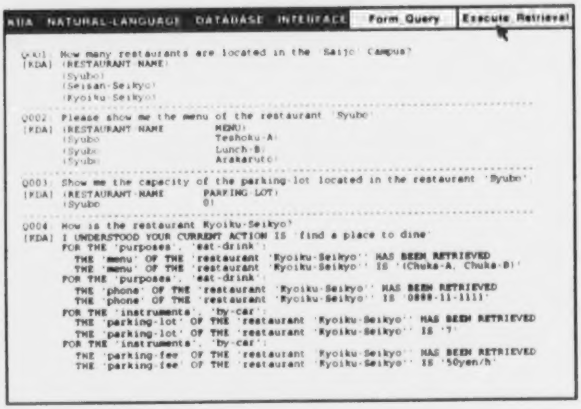
2.1. Indirect Database Queries and Informative Answers

A database query, whether it is expressed in a natural language or a formal language, can be considered internally as a specification of an *entity-selecting condition* which can be decomposed at least into three constructive factors (Cuppens et al., 1988):

entity-selecting condition ::= (*Subject-Entities*, *Retrieved-Attributes*, *Condition*),

where *Subject-Entities* indicates the target entity type in which the user is interested, *Retrieved-Attributes*, the attributes of *Subject-Entities* whose values are expected in the answer, and *Condition*, the qualification which *Subject Entities* must satisfy when *Retrieved-Attributes* is looked up. From this definition, we can classify the database queries into two categories: *direct queries* and *indirect queries*, by the criterion of whether or not the factor *Retrieved-Attributes* is specified explicitly in a database query. Direct query is query where *Retrieved-Attributes* are specified explicitly in the query expression. However, the user would sometimes expect to be permitted to input a terse query in which even the retrieved-attributes factor was not specified directly, but it can be inferred according to the contextual dialogue information and domain knowledge. In the following discussion, we call these kinds of queries *indirect queries*.

Figure 1 shows an example of using the indirect query facility in information retrieval. In the Q001 query, the user retrieved the names of restaurants located on the 'Saijo' campus. In the second query, the user chose a particular restaurant from the retrieval result of Q001 and further inquired into the menu of the restaurant. From the queries Q001 and Q002, the user's intended action that motivated the queries, i.e., *go to a restaurant to have a meal*, could be conjectured. The user's third query Q003 furnished additional information, i.e., the user inquired into the parking lot of the restaurant, for refining the inference about the user's action. As a result, the user's intended action could be conjectured as "*go to a restaurant by car to have a meal*". The retrieval result of the query Q003 showed that the restaurant 'syubo' did not have parking lot and therefore did not meet the requirement of the user. Instead of repeating the queries like Q002 and Q003 to retrieve information about another restaurant which satisfies the user's intended action, the user can input an indirect query like Q004 to retrieve such information.



```
Q001  How many restaurants are located in the 'Saijo' Campus?
[KDA] (RESTAURANT NAME)
      (Syubo)
      (Seisan-Seikyo)
      (Kyoiku-Seikyo)
-----------------------------------------------------------------
Q002  Please show me the menu of the restaurant 'Syubo'
[KDA] (RESTAURANT NAME          MENU)
      (Syubo                    Teshoku-A)
      (Syubo                    Lunch-B)
      (Syubo                    Arakaruto)
-----------------------------------------------------------------
Q003  Show me the capacity of the parking-lot located in the restaurant 'Syubo'
[KDA] (RESTAURANT NAME       PARKING LOT)
      (Syubo                 0)
-----------------------------------------------------------------
Q004  How is the restaurant Kyoiku-Seikyo?
[KDA] I UNDERSTOOD YOUR CURRENT ACTION IS 'find a place to dine'
      FOR THE 'purposes', 'eat-drink':
        THE 'menu' OF THE 'restaurant 'Kyoiku-Seikyo'' HAS BEEN RETRIEVED
        THE 'menu' OF THE 'restaurant 'Kyoiku-Seikyo'' IS '(Chuka-A, Chuka-B)'
      FOR THE 'purposes', 'eat-drink':
        THE 'phone' OF THE 'restaurant 'Kyoiku-Seikyo'' HAS BEEN RETRIEVED
        THE 'phone' OF THE 'restaurant 'Kyoiku-Seikyo'' IS '0888-11-1111'
      FOR THE 'instruments', 'by-car':
        THE 'parking-lot' OF THE 'restaurant 'Kyoiku-Seikyo'' HAS BEEN RETRIEVED
        THE 'parking-lot' OF THE 'restaurant 'Kyoiku-Seikyo'' IS '7'
      FOR THE 'instruments', 'by-car':
        THE 'parking-fee' OF THE 'restaurant 'Kyoiku-Seikyo'' HAS BEEN RETRIEVED
        THE 'parking-fee' OF THE 'restaurant 'Kyoiku-Seikyo'' IS '50yen/h'
```

Figure 1. An example of indirect information request.

Q004 is not a well defined query and cannot be interpreted directly by the standard database management system. In order to generate an informative answer to the indirect query Q004, the system must infer what type of information the user expects and reformulate the query Q004 into direct queries. For this purpose, the system must first determine the retrieved attributes that meet the user's requirement. An important key to determine the retrieved attributes is the user's intended action that motivated the indirect queries. The desirable retrieved attributes must be those that are useful for the user to achieve his intended action. From the example, the user's intended action was determined to be, "*go to a restaurant by car to have a meal*" from the contextual queries, Q001, Q002 and Q003, and therefore the retrieved attributes of the query Q004 can be determined to be the attributes, such as 'parking-lot', 'menu', 'phone', etc., for achieving the action.

2.2. A Knowledge-Base for Analyzing Indirect Queries

1) Semantic knowledge about databases

The semantic knowledge of the database plays an important role in interpreting natural language queries into formal database queries. In this system, the semantic

BEST COPY AVAILABLE

knowledge of the database is used in determining the retrieved attributes of indirect queries and reformulating the indirect queries into direct queries. By means of a semantic net diagram like KL-ONE (Brachman and Schmolze, 1985), an example of the semantic level description of a database can be drawn as shown in Figure 2.

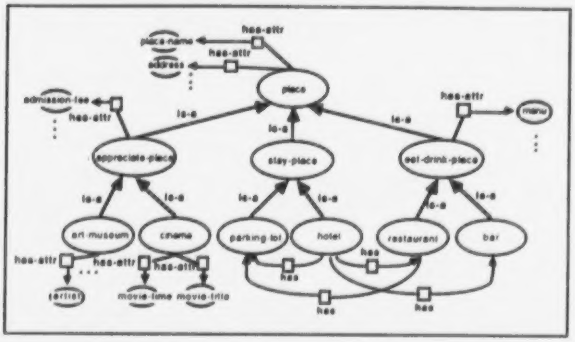

has-attr
place-name
has-attr
address
place
admission-fee
has-attr
is-a
is-a
is-a
has-attr
menu
appreciate-place
stay-place
eat-drink-place
is-a
is-a
is-a
is-a
is-a
is-a
art-museum
cinema
parking lot
hotel
restaurant
bar
has-attr
has-attr
has-attr
has
has
artist
movie-time
movie-title
has
has

Figure 2. A Semantic Network rpresenting data semantics.

In the semantic network the concepts, such as restaurant, hotel, etc., are depicted by ellipses. The attributes of each concept are represented by the link 'has-attr' (has-attribute) associated with it. The semantic relationships existing between the concepts are represented by the domain dependent semantic links, called SEMA-links. Each SEMA-link is named by a verb or a verb phrase, such as 'has' in this example. In Figure 2, the 'has-attr' link and the SEMA-links are depicted by squares with the link names. The concepts of a database application domain are organized into a *hierarchy* in which a concept can be connected to its super concept by the 'is-a' link which is depicted as a bold arrow.

2) Knowledge about user actions

The analysis of indirect queries requires the knowledge of user actions. This kind of knowledge would include the knowledge that supports the inference of a user's intended actions from the contextual queries and the knowledge used for determining retrieved attributes of indirect queries. A user's action model for the discourse domain is required to support the inference.

As shown in Figure 3, the user's action model consists of four parts: the 'Actions' part, the 'Objects' part, the 'Purposes' part and the 'Instruments' part. In the 'Actions' part, the knowledge of user actions is represented as a structured hierarchical network in which the higher level *action nodes* represent more general user actions, and lower level ones represent specific user actions. An action node can be connected to several specific action nodes with '*specialization*' links, called S-links for short, shown in Figure 3. The transition from a general action node to a more specific action node along the S-link is controlled on the basis of the '*Key-Concepts*' table associated with each action node. Figure 4. shows a simplified example of the 'Key-Concepts' tables. For each action node, except the 'General Action' node, a set of 'concepts' is defined with the 'Weight' value in the 'Key-Concepts' table of the action node. The 'Weight' value associated with each *concept indicates the importance of* the concept for recognizing the user action corresponding to the table. For example, the concept 'restaurant' is a very important key for recognizing the user action like "*find a place to dine*". Therefore, the highest 'Weight' value '1' was assigned to the concept 'restaurant' in the 'Key-Concepts' table of the action node '*Find a Place to Dine*'.

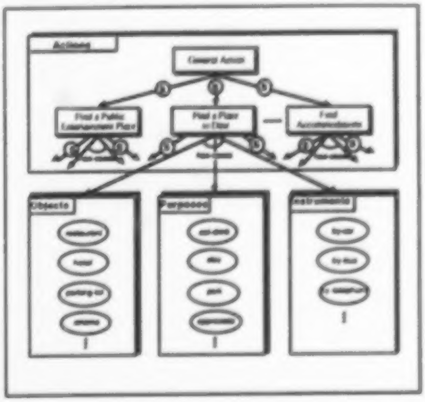
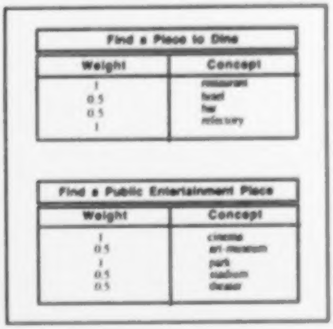


Figure 3. An example of user-actions model.



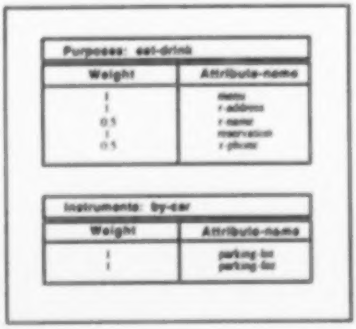
Find a Place to Dine

Weight	Concept
1	restaurant
0.5	hotel
0.5	bar
1	refectory

Find a Public Entertainment Place

Weight	Concept
1	cinema
0.5	art-museum
1	park
0.5	stadium
0.5	theater

Figure 4. Examples of Key-Concepts table.



Purposes: eat-drink

Weight	Attribute-name
1	menu
1	r-address
0.5	r-name
1	reservation
0.5	r-phone

Instruments: by-car

Weight	Attribute-name
1	parking-lot
1	parking-fee

Figure 5. Examples of Key-Attributes table.

In the action model, as shown in Figure 3, the potential case fillers of each action node are represented in the 'Objects' part, the 'Purposes' part and the 'Instruments' part, respectively. The potential fillers of the 'objects' case are represented in the 'Object' part of the model. For example, the concept 'restaurant' was defined as a potential filler of the 'object' case of the action node 'Find a Place to Dine' and then was linked to the action node by a 'has-cases' link. Another case of action nodes is the 'purpose' case whose potential fillers were represented in the 'Purposes' part of the action model. For example, when a user intends to find a place to dine a potential

EST COPY AVAILABLE

purpose of the user may be 'eat-drink'. Therefore, 'eat-drink' can be defined as a potential filler of the 'Purposes' case and linked to the action nodes 'Find a Place to Dine' by the 'has-cases' link. The 'Instruments' part of the model represents the potential fillers of the 'instruments' case of user actions. For example, the node 'by-car' can be considered as a potential filler of the 'instruments' case of the action node 'Find Place to Dine'.

As in the 'Key-Concepts' table corresponding to action nodes, each case filler defined in the 'Purposes' part or the 'Instruments' part associated with a 'Key-Attributes' table. The 'Key-Attributes' table defines which attributes are important for determining the case fillers. Figure 5 gives the examples of 'Key-Attributes' tables corresponding to the case nodes 'eat-drink' and 'by-car'. In the 'Key-Attributes' tables, the 'Weight' value assigned to each attribute-name indicates the salience of the attribute in determining the case filler.

3. USER-ACTIONS ANALYSIS

3.1. Inference of User's Beliefs

From a user's database query a set of user's beliefs about the database can be inferred. A user's belief can be regarded as an assumption that the user believes to be true about the content and the logical structure of a database. By limiting the discussion to the issue of database retrieval, we can classify the users' beliefs into the following four basic types (Wu and Ichikawa, 1993).

Type-I: Attribute-Value beliefs, **Bel**[instance (A, a)]
Type-II: Entity-Relation beliefs, **Bel**[relation (r, E1, E2)]
Type-III: Entity-Attribute beliefs, **Bel**[attribute-of (E, A)]
Type-IV: Entity-Instance Existence beliefs, **Bel**[existence (E, C)]

Type-I beliefs imply a type of user's assumption like 'a is a value of the attribute A.' For example, "Syubo' is a value of the attribute restaurant-name' is an instance of Attribute-Value beliefs and can be formulated as, **Bel**[instance (restaurant-name, Syubo)]. Type-II beliefs imply a type of user's assumption such as, 'there is a relation r between the entities E_1 and E_2.' For example, the belief, 'all hotels have a parking-lot,' can be classified into this type and can be formulated as, **Bel**[relation (has, hotel, parking-lot)]. Type-III beliefs imply a type of user's assumption such as, 'the entity-type E has an attribute A.' For example, 'all restaurants have the attribute menu' is an instance of Entity-Attribute beliefs and can be formulated as, **Bel**[attribute-of (restaurant, menu)]. Type-IV beliefs imply a type of user's assumption such as, 'the instances of the entity E which satisfy the condition C are existent." For example, 'there is a restaurant whose capacity is large than 300' is an instance of Entity-Instance Existence beliefs and can be formulated as, **Bel**[existence (restaurant(R), ((R(capacity) ≥ 300))].

From a user's query, a set of primitive beliefs as defined above can be formulated by the 'user-beliefs inference' program. For example, given the query, 'how many restaurants are located on the 'Saijo' campus?', two primitive beliefs can be inferred from the query as follows:

Bel[instance (campus, Saijo)]
Bel[relation (locate-in, restaurant, campus)].

3.2. Inference of a User's Intended Actions

As discussed in section 3.1, a set of user-beliefs can be inferred from a user's query. The inference of the user's intended action is carried out on the basis of the information collected about the user's beliefs and the action model. The inference of user actions is a process of translating the user-actions hierarchy from general action nodes to specific action nodes. The initial action node of the transition process is the root node of a user-actions hierarchy, for example, the 'General Action' node in Figure 3. When a user inputs a query, the user-beliefs analysis program is first activated and the information about the user's beliefs is derived from the user's query. Then, the user-action analysis program is activated to control the transition from 'General Action' to a specific action node that indicates the user's current action.

In the first step of user-actions analysis, the program generates a set of concepts, called SE-set (Subject Entities Set), which consists of the concepts derived from the user-beliefs description. Given a set of user-beliefs, the SE-set can be created according to the following rules:

(i) Given a Type-II belief description, **Bel**[relation (r, E_1, E_2)], the concepts E_1 and E_2 are defined as elements of SE-set;

(ii) Given a Type-III belief description, **Bel**[attribute-of (E, A)], the concepts E is defined as an element of SE-set;

(iii) Given a Type-IV belief description, **Bel**[existence (E, C)], the concept E is defined as an element of SE-set.

In the second step, each concept of SE-set is assigned a weight value. The weight value assigned to a concept plays a role in distinguishing the case where the concept itself represents a Subject Entity and another case of that concept is only used to restrict the Subject Entity. For the concept which represents a Subject-Entity, the value '1' is assigned to it. In the case where a concept is used to restrict the Subject-Entity, the concept would be assigned with a smaller weight value such as '0.5'.

In the third step of user-action analysis, the program compares the SE-set generated from a query with the Key-Concepts tables associated with user-action nodes to determine which action is closest to the user's intention. The determination of a user's intended actions is based on the calculation of the weight value, $\mathbf{W}_{\mathbf{action}}$ of each action node. Given an action node, **a1**, which has the Key-Concepts table:

$$KC_{\mathbf{a1}} = \{<C_1\ WKC_1>, <C_2\ WKC_2>, \cdots, <C_n\ WKC_n>\},$$

and an SE-set:

$$\text{SE-set} = \{<C_1\ WSE_1>, <C_2\ WSE_2>, \cdots, <C_m\ WSE_m>\},$$

and the concepts, C_1, C_2, $\cdots$, C_k, appeared simultaneously in the $KC_{\mathbf{a1}}$ and the SE-set, the weight value of the action node, **a1**, can be calculated by the following formula:

$$\mathbf{W}_{\mathbf{a1}} = WKC_1 \times WSE_1 + WKC_2 \times WSE_2 + \cdots + WKC_k \times WSE_k.$$

3.3. Inference of Case Fillers of an Action Node

Once a specific action node is determined as the current action node, the user action analysis program tries to determine the case fillers for the current action by analyzing successive queries. The method of deriving case fillers from the user-belief descriptions is same as the inference of user-action which has been discussed in the section 3.2. But, the inference of case fillers is based on a set of attributes extracted from the successive queries, called RA-set (Retrieved Attributes Set). Given a description of user-beliefs, the RA-set can be generated from the Type-I and Type-III beliefs. The rules for generating RA-set are:

(i) Given a Type-I belief description, **Bel**[instance (A, a)], the attribute 'A' is defined as elements of RA-set;

(ii) Given a Type-III belief description, **Bel**[attribute-of (E, A)], the concepts 'A' is defined as an element of RA-set.

For the example of user-system interaction shown in Figure 1, through the analysis of Q001 the user's intended action has been determined to be 'Find a Place to Dine'. From the query Q002, i.e., 'Please show me the menu of the restaurant 'Syubo'', and the query Q003, i.e., 'Show me the parking-lot numbers of the restaurant 'Syubo'', a set of user-beliefs can be derived as follows:

Q002: { **Bel**[instance (restaurant-name, Syubo)],
Bel[attribute-of (restaurant, restaurant-name)],
Bel[attribute-of (restaurant, menu)]};
Q003: { **Bel**[instance (restaurant-name, Syubo)],
Bel[attribute-of (restaurant, restaurant-name)],
Bel[attribute-of (restaurant, parking-lot)]}.

From the user-beliefs description, the RA sets of the queries Q001 and Q002 can generated as follows:

RA-set(Q002) = {restaurant-name, menu};
RA-set(Q003) = {restaurant-name, parking-lot}.

Each attribute of the RA sets is also assigned with a weight value. For an attribute which represents a Retrieved-Attribute, the attribute is assigned the value '1'. If a attribute is used to restrict the Subject-Entity, the attribute is assigned with a smaller weight value, e.g., '0.5'. For the RA sets given above, the weight value can be defined as follows:

RA-set(Q002) = {<restaurant-name 0.5>, <menu 1>};
RA-set(Q003) = {<restaurant-name 0.5>, <parking-lot 1>}.

Using the same method as used for determining the action node, the determination of a definite case filler of the current action node is carried out on the basis of the calculation of the weight value, $\mathbf{W_{case}}$, of every potential case filler. The case filler with the largest weight value is selected to be the user's intended case filler of the current action. For example, the 'Purposes' case filler of the current action node, 'Find a Place to Dine', can be determined to be 'eat-drink' because the Key-Attributes table of 'eat-drink' matched with the RA set 'RA-set(Q002)' with the weight value '1'. From the RA set 'RA-set(Q003)', the 'Instruments' case filler of the current action node can be determined to be 'by-car'.

4. INDIRECT QUERY ANALYSIS

4.1. The Determination of Subject-Entities

When the system accepts an indirect query, the system first derives the user's interested entity type, i.e., Subject-Entity, from the 'Objects' case filler of the current user-action node. An attributes set, called S-set, is then generated as a set of attributes which contains all attributes relating the Subject-Entity. For an indirect query, a S-set can be simply generated by the following way:

(i) Determine the 'Objects' case filler of the current action node and define it to be the Subject-Entity of the current indirect query.

(ii) For each attribute of the Subject-Entity which is defined directly in the semantic network model of the database or is inherited from the ancestor concepts of the Subject-Entity, the attribute becomes an element of the S-set.

For example, as shown in Figure 1, Q004 is an indirect query which the user posed after the system responded a 'null' result to the query Q003. In the interpretation of Q004, the program supposes that the user is still performing the action derived from Q001, Q002, and Q003 and generates a response to the indirect query Q004 on the basis of the description of the user's action. In the first step, the program must determine the Subject Entity for reformulating the indirect query Q004. Since the 'Object' case filler of the current action is 'restaurant' and an instance of the entity 'restaurant' was given in the query Q004, i.e., 'Kyoiku-Seikyo', the Subject-Entity of Q004 can be determined to be "restaurant 'Kyoiku-Seikyo'". In the second step, the program generates the S-set which includes all attributes of the Subject-Entity "restaurant 'Kyoiku-Seikyo'" by searching the semantic level description of the database as shown in Figure 2. For this example, the generated S-set is:

S-set = {phone, fax, address, holiday, business-hours, menu, price, seat-number, reservation-reception-period, parking-lot, parking-fee},

where, the attributes 'restaurant-name', 'seat-number', 'reservation-reception-period' and 'parking-lot' are the attributes defined directly in the concept 'restaurant' and others inherited from the super-concepts of the concept 'restaurant'.

4.2. The Determination of Retrieved Attributes

The determination of Retrieved-Attributes for an indirect query is carried out by taking an intersection of the attribute sets, S-set and G-set. The intersection set of S-set and G-set is determined as Retrieved-Attributes of the indirect query. In order to give an intuitive explanation of the process, consider the example given in Figure 1. Reformulating query Q004 into a direct query, the indirect query analysis program has derived the attribute sets S-set and G-set from the current user-action description. The intersection set of the two attribute sets can be calculated as follows:

Retrieved-Attributes = { phone, address, holiday, business-hours, menu, reservation-reception-period, parking-lot, parking-fee}.

Although the user did not specify directly which information about the restaurant 'Kyoiku-Seikyo' was desired in the answer, the system has inferred that the user's action is to find a restaurant for eating and drinking from the user's previous queries, i.e., Q001 and Q001. From the query Q003, the system also detected that the user intended to go to a restaurant by a car. On the basis of this inferred information about the user's intention, the system decides that the answer to query Q004 should include information, such as 'phone' 'address' 'holiday', 'business-hours', 'menu', 'reservation-reception-period', 'parking-lot', and 'parking-fee', about the restaurant 'Kyoiku-Seikyo'.

The determined Retrieved-Attributes are then sent to SQL-Database Query Generator (Wu and Ichikawa, 1992) with the Subject-Entity and the Conditions for generating a SQL database query for searching database. An informative response is then generated by a response generating program.

5. CONCLUSION

In this paper, we described a knowledge-based system for generating informative responses to users' indirect queries. The most important feature of this system is that it can infer a user's intended actions from the user's contextual queries and to answer the user's indirect query more informatively on the basis of a user-action model. The user-action model has been developed on the basis of the semantic-network model S-Net. Another feature of the system is the rule-based method for generating both affirmative and negative responses to indirect queries. Though the development of the system is a part of the KDA study, the system can work independently as an intelligent front-end to a database query system.

Acknowledgments

The authors are members and collaborators of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centres of Excellence Program of the Government of Canada, the National Science and Engineering Research Council (NSERC), and the participation of PERCARN Associates Inc. We are grateful to Canadian Cable Labs Fund for their financial assistance.

References

T. Gaasterland, P. Godfrey and J. Minker, (1992), "An Overview of Cooperative Answering," *J. of Intelligent Information Systems*, Vol: 1, pp. 123-157.

W. W. Chu and Q. Chen, (1992), "Neighborhood and Associative Query Answering," *J. Intelligent Information Systems*, Vol: 1, pp. 355-382.

S. J. Kaplan, (1982), "Cooperative Responses From a Portable Natural Language Query System," *Artificial Intelligence*, Vol: 19, pp. 165-187.

M. Kao, N. Cerone, and W. Luke, (1988), "Providing Quality Responses with Natural Language Interfaces: The null Value Problem," *IEEE Trans. on Software Engineering*, Vol: 14, pp. 959-984.

F. Cuppens, R. Demolombe, (1988), "Cooperative Answering: a Methodology to Provide Intelli-gence Access to Database," *Proc. Second Int. Conf. on Expert Database Systems*, pp. 333-353.

X. Wu and T. Ichikawa, (1992), "KDA: A Knowledge-based Database Assistant with A Query-Guiding Facility," *IEEE Trans. on Knowledge and Data Eng.*, Vol: 4, pp. 443-453.

H. Kautz and A. Allen, (1986), "Generalized plan recognition," *Proc. National Conf. on Artificial Intelligence*, pp. 32-37.

M. Vilain, (1990), "Getting Serious about Parsing Plans: a Grammatical Analysis of Plan Recognition," *Proc. National Conf. on Artificial Intelligence*, pp. 190-197.

P. Beek and R. Cohen, (1991), "Resolving Plan Ambiguity for Cooperative Responses Generation," *Proc. 12th Int. Joint. Conf. on Artificial Intelligence (IJCAI-91)*, pp. 938-944.

R. J. Brachman, and J. G. Schmolze, (1985), "An Overview of the KL-ONE Knowledge-Repre-sentation System," *Cognitive Science*, Vol: 9, pp. 171-216.

X. Wu and T. Ichikawa, (1993), "A Knowledge-based System for Correcting Users' Misconcep-tions in Database Retrieval," *Int'l J. of Software Eng. and Knowledge Eng.*, Vol: 3, pp. 115-138.

An Experimental Comparison of Object-Orientation and Functional-Decomposition as Paradigms for Communicating System Functionality to Users

Tony Moynihan

School of Computer Applications, Dublin City University, Dublin 9, Ireland

Abstract

This paper describes an experiment to evaluate the relative effectiveness of functional-decomposition and object-orientation as paradigms for client/developer communication in the early stages of the system development process. The subjects were twenty executives attending a management development program. The experimental task required the subjects to comment critically on the content and format of two analyses of equivalent content. The first analysis took the form of a functional-decomposition. The second analysis took the form of an object-model. The results suggest that functional-decomposition is the more effective of the two paradigms as a vehicle for early client/developer communication. Also, the subjects judged the functional-decomposition to be superior to the object-model on a number of important attributes.

1. INTRODUCTION

The Object-Oriented approach is well established in programming. Recently, its application has been extended to include systems analysis and design (see, for example, Coad and Yourdon (1991) and Rumbaugh et al.(1991)). Simultaneously, approaches to software development which emphasise functional decomposition are increasingly seen to be 'old fashioned.'

One of the major 'selling-features' of the O-O approach is its use of a uniform set of concepts across the development process , thus supporting verification, traceability and re-use. Presumably, to obtain the full benefits of this uniformity of representation , the O-O approach should be used from the very start of the development process i.e for requirements elicitation and validation. But SHOULD it ? How effective is the O-O paradigm as the main vehicle of communication between the typical 'customer' and the developer? Are object-oriented descriptions more (or less) effective as a basis for user-validation than are descriptions based on traditional functional decomposition? The experiment described below was designed to help answer these questions.

The subjects were twenty business managers attending a part-time graduate program at the Irish Management Institute. The scenario painted for the subjects was one in which two (ficticious) computer consultants had been asked to independently report on how IT systems could help run the 'Happy Hours Health-Farm' , also ficticious!. The consultants' preliminary analyses of the health farm were given to the subjects. One analysis was structured as a functional-decomposition. The other as an object-model. The contents of the two 'analyses' were equivalent, although this fact was not made explicit to the subjects.

Subjects were asked to critique the content of each of the two analyses. Subjects were also asked to record their opinions on the relative merits of the two paradigms. The 'effectiveness' of the two paradigms in communicating system functionality was inferred from the 'quality' of the subjects' critiques and from the opinions they expressed about each paradigm.

2. THE 'SCENARIO' PAINTED FOR THE SUBJECTS

The first step was to choose a suitable application domain as a vehicle for the experiment. An application domain which was very unfamiliar to the subjects would clearly be a bad choice. Subjects would have no basis on which to identify missing features or other anomalies in the experimental materials. An application domain which was very familiar to the subjects would also be a bad choice. A subject might not mention an apparent oversight in the material on the assumption that it was so obvious a point that it had been deliberately excluded.

Guided by these considerations, the application domain chosen was the management of operations in a fictitious organisation called the 'Happy Hours Health-Farm.' Obviously, few subjects were likely to have experienced a stay at a health-farm. But I felt most would be familiar with the broad theme, if only through novels and films.

I created a 'story-line' involving two computer consultants, Gerry Kelly and Liam Ryan. In the story-line, Gerry and Liam have each been asked by health-farm management to independently give a view on how IT systems could support health-farm operations. Both have made initial visits to the health-farm and have completed their preliminary analyses. These 'analyses' , Kelly's in the form of a functional-decomposition and Ryan's in the form of an object-model, were the basis for the experimental task.

3. THE EXPERIMENTAL MATERIALS

When constructing the consultants' 'analyses', I kept in mind the need for the content of the two analyses to be 'equivalent' , in the sense that the substantive content of either should be derivable from the content of the other. This point is central to the logic of our experiment ; the goal was to compare the effectiveness of the two paradigms as information presentation styles in the user-validation process. Thus equivalence of content of the two 'analyses' was obviously essential.

A second concern was the need for the two analyses to be at the same level of abstraction. Barros(1992) makes the point that user-oriented system specifications typically contain a mixture of user-goals (the 'why'), the functions to be provided (the 'what'), and implementation structures (the 'how'). Given the goal of the experiment, I chose to restrict the analyses to the functional level of abstraction (the 'what').

My next task was to choose exemplars of the two paradigms. In other words, to select the specific representation schemes to be used in the experiment. For the O-O paradigm, I chose Rumbaugh's Object Model Notation (Rumbaugh et al.,1991) , as it appears to have achieved a significant level of industry acceptance. The notation supports Generalisation (Inheritance) , Aggregation, Class Attributes/ Operations and various types of Class Association.

Functional-Decomposition is a universally used, well understood paradigm. Many schemas for presenting a functional decomposition have been proposed over the years. These differ only in minor points of layout. The schema I chose is that suggested in Martin(1982).

The 'analyses' which I constructed, one for each 'consultant', are shown in appendices 1 and 2. In each case, I prefaced the analysis with context-setting comments and explanation. With a view to achieving equivalence of content of the two analyses, I excluded class attributes and class associations from the O-O analysis. Given that the analyses are ostensibly 'first-cut' attempts by the two consultants, this restriction is probably not unrealistic.

Having constructed the two analyses, I next created 'An Extract from the Health-Farm Prospectus.' This 'extract' was given to the subjects and was described as a source of background information on the health-farm.

Many of the 'facts' provided in the extract were reflected in the consultants' analyses. However, the extract was 'seeded' with a number of **additional** 'facts' of potential relevance, which I deliberately omitted from the consultants' analyses. The frequency with which the subjects detected these apparent 'oversights' in the consultants' analyses was one of the dependent variables in the experiment. Five such 'seeds' were deliberately sown :

(i) The existence of a golf-course, horse-riding school, indoor swimming pool and hydro-massage tanks for use by guests;
(ii) The employment by the health-farm of psychologists, therapists and beauty experts;
(iii) The organisation of activities such as music-recitals, painting sessions and mini-bus excursions;
(iv) The availability for hire by guests of specialist equipment such as ultra-sound deep-tissue massage canopies.
(v) The concept of a personalised 'inner-health' programme for each guest.

Finally, I designed a simple questionnaire to capture the subjects' views on the relative merits of the two 'styles' of presentation.

4. THE SUBJECTS

The subjects were twenty middle and senior managers from a variety of Irish organisations. All were students on a part-time executive masters program in organisational science at the Irish Management Institute. Their mean age was about 35 years. None were information systems specialists. Their course included no technical material on IS.

5. THE RESULTS

The goal of the experiment was to compare the effectiveness of Object-Orientation and Functional-Decomposition as paradigms for communicating system functionality to users. The notion of 'effectiveness' was operationalised in terms of subjects' task-performance and in terms of subjects' opinions on the two 'styles' of analysis.

5.1 Subjects' task-performance

A subject's task-performance was measured in two ways :

(i) by the number of 'seeds' the subject mentioned as being missing from the analysis. It will be recalled that a 'seed' is a potentially important 'fact' planted in the prospectus extract and deliberately omitted from the consultants' analyses. Five 'seeds' were sown.

(ii) by the 'quality' of the subject's critique of the **content** of the analysis. I did not define 'quality' in advance of running the experiment. I decided to adopt an inductive approach by letting the data 'speak for itself.'

Table 1 contains a breakdown of the twenty subjects by number of missing 'seeds' identified. There is no evidence in Table 1 to suggest that seed detection rate differed across the two analyses.

	Detected No Seeds	Detected One Seed	Detected Two or More Seeds	Total
Functional Decomposition	12	3	5	20
Object-Model	15	3	2	20

Table 1 : Numbers of subjects detecting missing 'seeds'

Subjects' comments on the **content** of each analysis fell into one of two categories.The first category contained references to apparent omissions in the analysis (other than the 'seeds'). Examples of comments in this category include :

"The purchasing activity is totally missing."
"What about stock control for the kitchens and bars?"
"Scheduling of maintenance must be important"

The second category contained comments of a more strategic nature. Comments in this category related to the perspective taken in the analysis, to the relative emphasis given to the different components in the analysis, or to the priorities implied by the analysis. Examples of comments include :

"Surely the priority should be on urgent financial reporting systems?"
"The goal seems to be to maximise the use of IT. Its a bit mindless."
"Should be more emphasis on measuring the cost-effectiveness of people and facilities."
"Emphasis on **systems** to schedule appointments etc. doesn't square with the personal attention promised in the prospectus."

Tables 2 and 3 below give breakdowns of the twenty subjects by the numbers of comments made in each of these two categories for each of the two analyses.

	Identified None	Identified One or Two	Identified Three or More	Total
Functional Decomposition	6	9	5	20
Object-Model	6	11	3	20

Table 2 : Numbers of subjects identifying missing/needed extra features

	Made no Comment	Made one or More Comments	Total
Functional Decomposition	9	11	20
Object-Model	15	5	20

Table 3 : Numbers of subjects making comment on overall perspective/balance/priority

There is no evidence in Table 2 to suggest that either representation was more successful than the other in terms of the frequency of comments elicited about missing/needed extra features. But what of the **substance** of these comments? For example, did the object-model elicit more 'object oriented' comments than did the functional-decomposition? A content analysis of the comments showed that this was not the case. No systematic differences in the focus of comments relating to missing / needed extra features were found.

Regarding subjects' comments in the second category, Table 3 suggests that the functional-decomposition was far more effective than the object-model in eliciting comments of a strategic nature ($p < .05$ on Fisher's Exact Test.) Again, a content analysis revealed no differences in focus between the 'strategic' comments elicited by the object-model and those elicited by the functional-decomposition.

5.2 Subjects' opinions on the two 'styles' of analysis

Subjects' opinions on the relative merits of the two **styles** of analysis were obtained by means of the post-experiment questionnaire.

Twelve subjects said they preferred the functional-decomposition. Eight subjects said they preferred the object-model. A content analysis of the completed questionnaires showed that most of the subjects drew from the same set of six broad criteria in comparing the two analyses. In Table 4 below, I have expressed these criteria in the form of the six desireable attributes that the subjects believed an analysis style should display. The table shows the numbers of subjects making positive and negative comments about each of the two styles on each of the six attributes.

		Making Positive Comment	Making Negative Comment	Making No Comment	Total
The notation and concepts are easy to understand	Functional-Decomposition	15	1	4	20
	Object-Model	5	11	4	20
It helps the reader to detect incompleteness and internal inconsistency	Functional-Decomposition	4	0	16	20
	Object-Model	1	2	17	20
It provokes the reader to make comments and ask questions	Functional-Decomposition	8	0	12	20
	Object-Model	0	1	19	20

It gives the reader a holistic understanding of the application domain	Functional-Decomposition	6	0	14	20
	Object-Model	2	6	12	20
It helps the reader to evaluate likely implementation benefits and priorities	Functional-Decomposition	9	0	11	20
	Object-Model	0	7	13	20
It helps the reader to visualise an implementation of the system	Functional-Decomposition	3	6	11	20
	Object-Model	6	5	9	20

Table 4 : Numbers of subjects making positive and negative comment on the **style** of each analysis

Table 4 shows that the subjects found the functional-decomposition easier to understand than the object-model (p<.05 on Fisher's Exact Test). Many said that functional-decomposition was a familiar and intuitively obvious concept, but found the concept of an object-model to be new and somewhat inscrutable. Interestingly, this 'novelty' does not appear to have detracted from the subjects' performance in detecting missing 'seeds' and other features in the object-model (see tables 1 and 2).

The second attribute on which the subjects judged the two analysis styles was the extent to which each helped the subject to detect incompleteness and internal inconsistency in the analysis. Table 4 provides no significant evidence of an overall preference either way on this attribute. This finding is consistent with the absence of any statistically significant difference between the subjects' 'seed' detection rates on the two analyses.

The third attribute was the extent to which the style 'provoked' the subjects to ask questions or make comments about the content of the analysis. Table 4 shows that the functional-decomposition was seen by the subjects to be the more effective in this regard (p<.05 on Fisher's Exact Test). As one subject put it : "Gerry's analysis is nice and open and warm. It invites comment. But Liam's is tight and closed and clinical. It's hard to get excited about it."

On the fourth attribute, the extent to which the subjects believed the analysis gave a complete and integrated (as opposed to a partial and fragmented) perspective of the business, there is significant evidence that the subjects thought the functional-decomposition to be the more effective (p<.05 on Fisher's Exact Test). This finding must be interpreted with care. Factually, the content of the two analyses was identical.

So, it seems that the functional-decomposition gave the subjects a stronger **perception** of 'wholeness' than did the object-model.

The fifth attribute was the extent to which the analysis helped the subjects to anticipate possible implementation benefits and priorities. Overall, they clearly felt that the functional-decomposition was the more effective in this respect ($p<.05$ on Fisher's Exact Test). This finding, and the two findings immediately above, are consistent with the earlier finding that subjects made more comments of a 'strategic' nature about the content of the functional-decomposition than about that of the object-model.

The sixth attribute which the subjects used for comparison purposes was the extent to which the analysis helped them to visualise, in concrete terms, a possible implementation of the system. There is no statistically significant evidence in table 4 to suggest that the subjects believed one analysis to predominate over the other in this respect.

6. Conclusions

Regarding task performance, there is no evidence that subjects' rates of detection of missing 'seeds' differed across the two analyses. Nor is there evidence to suggest that either analysis was more successful than the other in eliciting criticisms and suggestions of a **detailed** nature. However, the functional-decomposition was by far the more successful in eliciting comments of a more **strategic** nature about the business and its needs, and about apparent 'mis-matches' between these needs and the content of the analysis.

Overall, subjects said that, compared with the object-model, the functional-decomposition was :

- easier to understand;
- provoked them to ask more questions and to make more comments;
- gave them a more holistic understanding of the business;
- better helped them to evaluate likely implementation benefits and priorities.

Obviously, the findings of an experiment such as this can be subject to many threats to their validity. In this particular case, threats could include the artificiality of the experimental task, possible deficiencies in the operationalisation of the two paradigms, and defects in the experimental materials.

On the assumption that the findings have some external validity, what are the implications of the findings for research and practice? It seems that object-orientation, at least in the form of a 'naked' object-model as used in this experiment, may not be effective as the primary vehicle for communication between client and developer at the earliest stages of system development. It appears that 'old-fashioned' functional-decomposition may be more appropriate.

But today, the O-O paradigm is demonstrably 'delivering' many benefits at the later stages of the system development process. So, the challenge seems to be to create a new synthesis of the two paradigms, which exploits the strengths of both, for use at the all-important 'front-end' of the development process.

References

Barros, P.A. (1992), "The nature of bias and defects in the software specification process", CS-TR-2822, Computer Science Technical Report Series, Univ. of Maryland.

Coad, P. and E.Yourdon, (1991),"Object-Oriented Analysis", Yourdon Press, New Jersey.

Martin J. (1992), "Strategic Data Modelling", Prentice-Hall, New Jersey.

Rumbaugh J. et al, (1991), "Object-Oriented Modelling and Design",
Prentice-Hall, New-Jersey.

1. **Resource Scheduling**
   - 1.1 Schedule On-Arrival, Mid-Course and On-departure Guest Health-Checkups
   - 1.2 Process Guest Bookings for Saunas, Sunbeds and Exercise Machines
   - 1.3 Produce Weekly Duty Roster for every Bedroom Cleaner, Laboratory Technician and Keep-Fit Facility Attendant

2. **Resource Allocation**
   - 2.1 Assign Doctors, Nurses and Physiotherapists to Guest Health-Checkups
   - 2.2 Assign Doctors, Nurses and Physiotherapists to Guest Personal Support Teams (one of each to a team)
   - 2.3 Assign Bedrooms to Guests and Employees

3. **Formulate Guest Personal Health Programmes**
   - 3.1 Formulate Guest Personal Menus
   - 3.2 Formulate Guest Personal Exercise Programmes

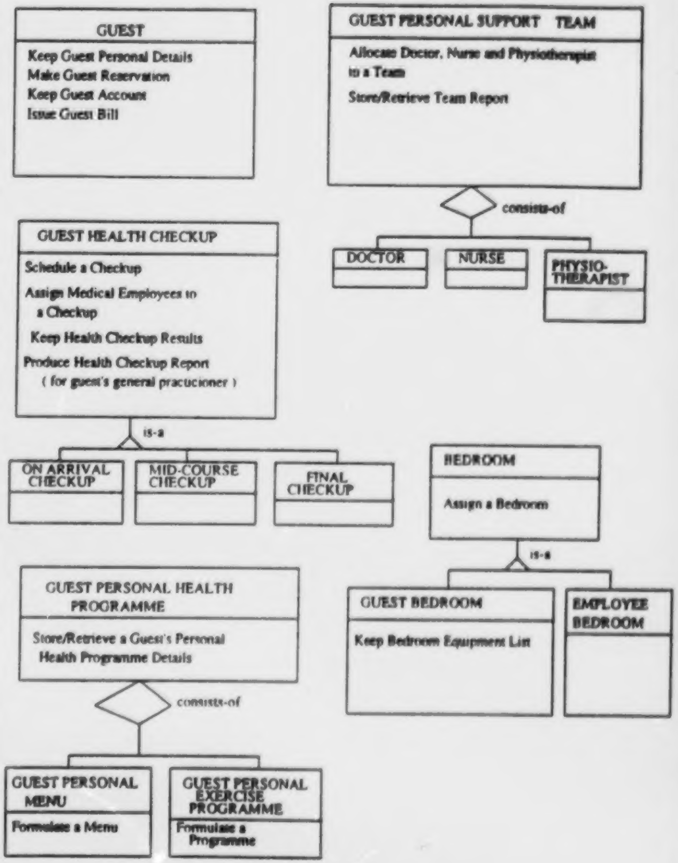
4. **Store/Retrieve Guest Records**
   - 4.1 Guest Personal Details
   - 4.2 Guest Health-Checkup Results
   - 4.3 Guest Personal Support-Team Reports
   - 4.4 Guest Personal Health Programme Details
   - 4.5 Guest Keep-Fit Facility Usage Details (details of usage for each guest)

5. **Equipment Maintenance and Inventories**
   - 5.1 Keep the Problem/Incident Log for each Sauna, Sunbed and Exercise Machine
   - 5.2 Keep Maintenance Log for each Sauna, Sunbed and Exercise Machine
   - 5.3 Keep Guest Bedroom Equipment List (shows equipment in each guest bedroom eg. infrared lamps, massage pads.)

6. **External Reporting**
   - 6.1 Produce Health-Checkup Reports for Guests' General Practicioners (these are based on the guests' health-checkup results)

7. **Accounting and Payroll**
   - 7.1 Guest Accounting
     - 7.1.1 Make Guest Reservations
     - 7.1.2 Maintain Guest Accounts (including Billing)
   - 7.2 Payroll and Employee Records Administration

Appendix 1 : Gerry Kelly's Analysis


GUEST
Keep Guest Personal Details
Make Guest Reservation
Keep Guest Account
Issue Guest Bill
GUEST PERSONAL SUPPORT TEAM
Allocate Doctor, Nurse and Physiotherapist to a Team
Store/Retrieve Team Report
consists-of
DOCTOR
NURSE
PHYSIO-THERAPIST
GUEST HEALTH CHECKUP
Schedule a Checkup
Assign Medical Employees to a Checkup
Keep Health Checkup Results
Produce Health Checkup Report (for guest's general practicioner)
is-a
ON ARRIVAL CHECKUP
MID-COURSE CHECKUP
FINAL CHECKUP
BEDROOM
Assign a Bedroom
is-a
GUEST BEDROOM
Keep Bedroom Equipment List
EMPLOYEE BEDROOM
GUEST PERSONAL HEALTH PROGRAMME
Store/Retrieve a Guest's Personal Health Programme Details
consists-of
GUEST PERSONAL MENU
Formulate a Menu
GUEST PERSONAL EXERCISE PROGRAMME
Formulate a Programme

Appendix 2 : Liam Ryan's Analysis (extract from)

BEST COPY AVAILABLE

COMPLEXITY REDUCTION AND MODELLING TRANSPARENCY IN CASE TOOLS

Sjaak Brinkkemper, Rolf Engmann, Frank Harmsen and Rob van de Weg

Department of Computer Science, University of Twente,
Postbus 217, NL-7500 AE Enschede, The Netherlands,
email: {sjbr, engmann, harmsen, vandeweg}@cs.utwente.nl

Abstract

Systems analysis usually is supported by several diagramming techniques provided by CASE tools in an integrated environment. In practical cases individual diagrams may be large and complex; another complicating factor is that different diagrams are interrelated because they portray the same or closely related concepts within different contexts, e.g. Entity-Relationship diagrams and Data Flow diagrams. The present paper describes the techniques that are used for reducing the complexity in diagramming techniques in commercially available CASE tools. Maintenance of compatibility between different diagrams, so-called modelling transparency, has also been evaluated for these CASE tools.

1. INTRODUCTION

In recent years an abundance of CASE tools supporting analysis, design and realization of information systems have become available. In the present study we confine ourselves to the so-called upper CASE tools, supporting the initial stages of systems development, especially systems analysis. Integrated CASE tools usually include several interrelated diagram editors, for instance for entity-relationship diagrams (ERDs), data flow diagrams (DFDs), state transition diagrams (STDs), structure charts (SCs), decomposition diagrams, etc. (Martin and McClure, 1988; Rothstein et al., 1993).

Information systems analysis usually yields a large number of data, due to the complexity of the information system to be developed. These data are represented by many diagrams, which may be of the same or of different types. The complexity and size of the intended information system is reflected in the diagrams and the relations between diagrams. For the developers management of such interrelated and complex diagrams without CASE tool support may be cumbersome and unwieldy.

The complexity of diagramming techniques can be divided into two main problems, viz. the complexity within diagrams, and the relationships between diagrams of different types, so-called modelling transparency (Brinkkemper, 1993). There exist several techniques for handling complexity in both situations. An overview of complexity reduction within diagrams is discussed in section 2; modelling transparency is discussed in section 3. Section 4 describes a formalization of some complexity reduction techniques. Practical results of investigations on eight commercially available CASE tools are discussed in section 5.

2. COMPLEXITY REDUCTION

In practice diagrams used in systems analysis contain a multitude of symbols, representing concepts recognized in the systems to be developed. A plethora of symbols, such as squares, rectangles and circles, combined with many relationships between symbols, represented by arrows and lines may impede instead of enhance understanding the diagram. The human mind can only grasp a small number of concepts simultaneously, because of the limited capacity of our short-term memory. Therefore, users should be supported by tools and techniques to survey and interpret complex diagrams.

Complexity reduction in diagrams will be defined in this paper as all techniques, available to users and designers to enhance surveyability and maintainability of diagrams containing many interrelated symbols.

Complexity reduction techniques are relevant in many problem solving strategies; see e.g. Martin and McClure (1988), and Ossher (1987). We distinguish the following approaches:

- Divide and conquer - a large problem area is divided into a group of smaller and relatively independent problem areas that are easier to understand and to solve. In diagramming this approach corresponds to viewing.
- Subject area - an area of specific interest is isolated from its context and is considered rather detached from its environment. Such an area is a limited view on a larger domain, taking specific aspects into consideration. In diagramming this approach corresponds to viewing, and colouring (or highlighting) of specific objects.
- Top down specification of a complex system showing more detail on the lower levels. A symbol on a specific level may be decomposed on a lower level, showing its component parts and connections between components in more detail. This kind of decomposition is also known as explosion.
- Stepwise refinement from concepts, relationships, and boundaries that are initially vague and imprecise, which become more distinct and accurate during the development process. This means that details that are not of immediate interest are not included in higher levels, but are added to lower levels without an equivalent at higher levels. In diagramming this corresponds to hiding.

The latter two approaches can be represented by a hierarchical decomposition. A complex model may be portrayed on several levels of abstraction, the highest level or root level providing a global view of the complete system, the lower levels more details or more precise information. Another term used in the literature for decomposition is downward levelling (Yourdon, 1989).

The correspondences and analogies between the complexity reduction techniques discussed in this section and the strategies mentioned in the previous are an indication of the relevance and completeness of the set of techniques.

Complexity reduction techniques can be divided into conceptual complexity reduction techniques and presentation or graphical techniques. Conceptual techniques are dependent upon the semantics of the diagrams, such as viewing, hiding, colouring, and decomposition. Graphical techniques are zooming, scrolling, off-page connectors, automatic layout and adaptable symbols. In commercially available CASE tools diagram editors usually offer several techniques to support complexity reduction.

2.1 Conceptual complexity reduction

Views

A complex diagram may be divided into several functional views, each providing a limited scope of the problem area; a view contains a subset of the whole according to a particular aspect. An example of different views that can be derived from the same diagram is a diagram in which both data flows and control flows are shown for the same set of processes. Such a diagram can be divided into a proper data flow diagram showing processes, data stores and data flows, and a control flow diagram showing the same processes and the control flows between those processes. Such control flow diagrams are particularly relevant for the design of real-time systems (Hatley and Pirbhai, 1987).

Well-known applications of the view concept are so-called external views or external schema's based on a conceptual schema in the ANSI/SPARC three-schema architecture. A graphical example of views in terms of ER-schema's is given by Elmasri and Navathe (1994), pages 461 - 462.

A third application of viewing is dialogue modelling by state transition diagrams, where different but interrelated dialogues are represented by views, e.g. dialogues for the normal case, the help dialogue, and the feedback dialogue (Koesen et al., 1989). Figure 1 shows a state transition diagram with two views above.

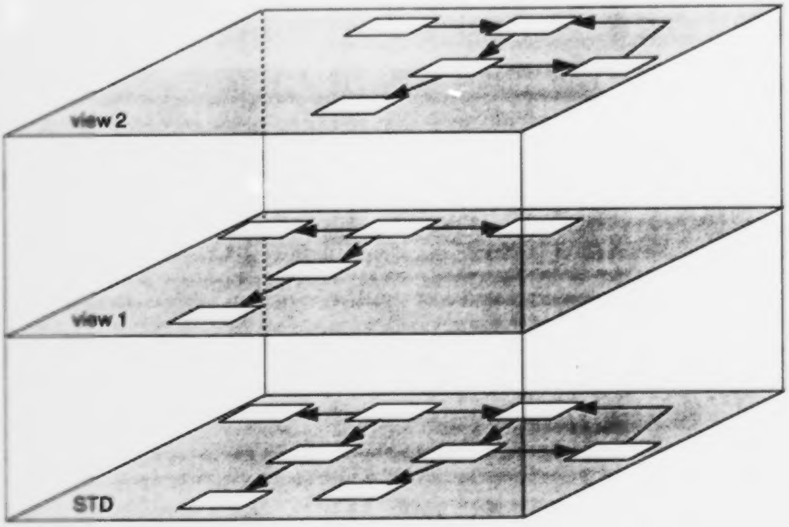


Figure 1. State transition diagram with views

Hiding

Surveyability of diagrams may be improved by omitting or hiding details. Some techniques for ERDs include attributes of the entities in the diagrams, e.g. Elmasri and Navathe (1994) and Coad and Yourdon (1990), which may yield very crowded diagrams. Omission of the attributes from such an ERD may increase clarity and readability. This technique is conceptually closely related to viewing. Hiding may be used for stepwise refinement.

Decomposition

Decomposition can be applied recursively, yielding a hierarchy of decomposition levels. Decomposition is often applied to data flow diagrams, starting with the main business processes on the root level or with a context diagram showing a single bubble representing the entire system interacting with external agents (also called terminators). In the case of decomposition in the sense of top down specification the subsequent decomposition levels should be consistent, such that the relations that an object has with its environment are retained on the lower level. Maintenance of consistency can be supported by the CASE tool used for drafting decomposed diagrams. In case of decomposition in the sense of stepwise refinement, maintenance of consistency should not be enforced strictly to allow uncertainty and inaccuracy on a more global level. For this reason the user of a CASE tool should have the options to enable and disable consistency maintenance.

Decomposition always yields diagrams of the same type, e.g. a data flow diagram can only be decomposed into (more detailed) data flow diagrams. Figure 2 shows decomposition of a symbol on the next-lower level of abstraction.

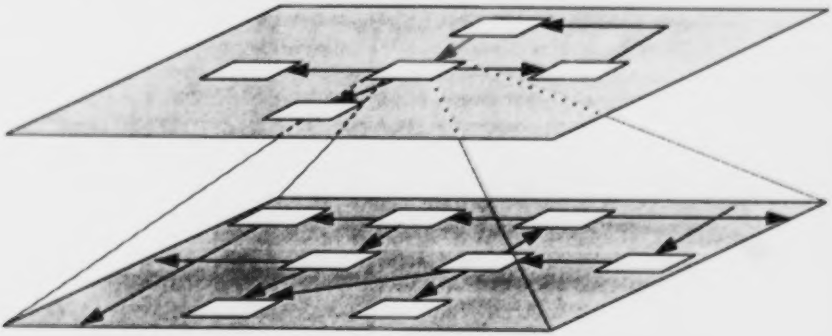


Figure 2. Decomposition

A special application of decomposition techniques is the use of so-called substitution nodes in hierarchical Petri nets (Huber et al., 1991). In such a Petri net a substitution node may be replaced by a more detailed Petri net. The detailed net may be parameterized, such that it can be substituted into several substitution nodes of a higher level (more global) Petri net.

Colouring

The use of colours in diagrams may improve surveyability by adding extra information, e.g. objects of the same type may have the same colour, or diagrams of the same type may have the same background colour, or the current menu or object has a specific, conspicuous colour. A variation on colouring is highlighting specific symbols; however, highlighting facilities were not provided by the CASE tools investigated.

2.2 Graphical complexity reduction

Zooming and scrolling

Zooming and scrolling are well-known facilities, usually provided in combination with windowing. By selecting a part of a diagram with the mouse or with a zoom box the selected part of the diagram can be enlarged by zooming to fill a whole window. The window can be scrolled up and down over the diagram with scroll bars and scroll buttons. Zooming and scrolling do not reduce complexity of the diagram as such, but support surveyability of diagrams.

Off-page connectors
Off-page connectors are special symbols to decrease the number of connections between symbols in the same diagram. An off-page connector is a uniquely identified symbol appearing twice in a diagram. It represents the connection between the symbols connected with the off-page connectors. In this way long and crossing lines may be avoided; they enable splitting large diagrams into separate partial diagrams. Clicking on a connector in a displayed subdiagram causes switching of the display to the connected subdiagram.

Automatic layout
Diagrams with many nodes connected by crossing lines may be simplified by an automatic layout function by displacing the symbols in the diagram, such that the number of crossing lines is minimized. In this way the surveyability of the diagram may be enhanced.

Adaptable symbols
Some CASE tools permit symbols to be adapted. This can be useful for maintaining corporate standards between departments in a company.

3. Modelling transparency

The many diagrams modelling an information system that are produced by an integrated CASE tool are highly interrelated, due to the fact that the same objects are represented in several of those diagrams, albeit in a different setting. CASE tools supporting integrated diagramming techniques should enable the creation, modification and removal of relationships among the diagrams and switching from one diagramming technique to another according to a particular dependency between the diagrams. Because of the resulting transparency of the diagrams and the diagramming techniques to each other, we call this functionality of a CASE tool *modelling transparency* (Brinkkemper, 1993) or simply *transparency* for short. Transparency indicates the absence of intermediate user operations to transfer from one diagram to another. The extent to which intermediate operations are superfluous in tools is called the *degree of (modelling) transparency*. The need for many intermediate user operations for switching between diagrams complicates the connections between diagrams and is assessed as a low degree of transparency. A CASE tool with a high degree of transparency supports the user in switching between diagram techniques in an inconspicuous and user-friendly way. In Brinkkemper (1993) four degrees of modelling transparency are distinguished:

0. Stand-alone CASE tools with a non-accessible repository do not support modelling transparency; their degree of transparency is 0. Developers are forced to recognize and monitor dependencies manually.
1. CASE tools with an open, accessible repository enable dependencies to be established. However, these links can only be shown by abandoning the present diagram editor, starting up the other CASE tool and displaying the other diagram. This corresponds to transparency degree 1.
2. A higher form of transparency is offered by CASE tools providing direct support in switching from one diagram to the other. The best result is achieved if both diagrams can be displayed simultaneously and changes to one diagram are immediately reflected in the other. This kind of linking diagrams corresponds to transparency degree 2.
3. Transparency degree 3 denotes CASE tools offering the so-called hypertext functionality. A user of such a CASE tool can freely define links between any object in one diagram to any object in another diagram. The related diagrams then become immediately accessible to each other, making it possible to traverse the dependen-

cies of the diagrams in an arbitrary, user-defined way.

In a tool with transparency degree 2 the links between diagrams have been defined within the CASE tool itself. The user can only use the links that have been predefined in the tools. In a CASE tool with transparency degree 3 the user is free to define any link whatsoever. Hypertext functionality offers more flexibility but might be only beneficial for very experienced and proficient developers, due to the difficulty of managing a network of more or less casual dependencies.

Editor transfer charts

The possibilities for transferring between several diagram editors within a CASE tool can be represented by so-called editor transfer charts (Brinkkemper, 1993). An editor transfer chart contains the diagram editors represented by bold rounded rectangles and labeled arrows representing the links to transfer from one diagram editor to another. The arrows represent the direct links that exist between the corresponding editors. The direction of an arrow represents the direction in which it is possible to transfer between the editors. The name of a specific component appears as the label on an arrow. An arrow with the name of a diagram component denotes the capability of the CASE tool to transfer from the one editor to the other by choosing any of the instances of the component, e.g. by a mouse click on a component of a diagram. Text editors and input forms are represented by smaller blocks with thin edges. Such an editor transfer chart provides a concise survey of the transfer functionality of a specific CASE tool.

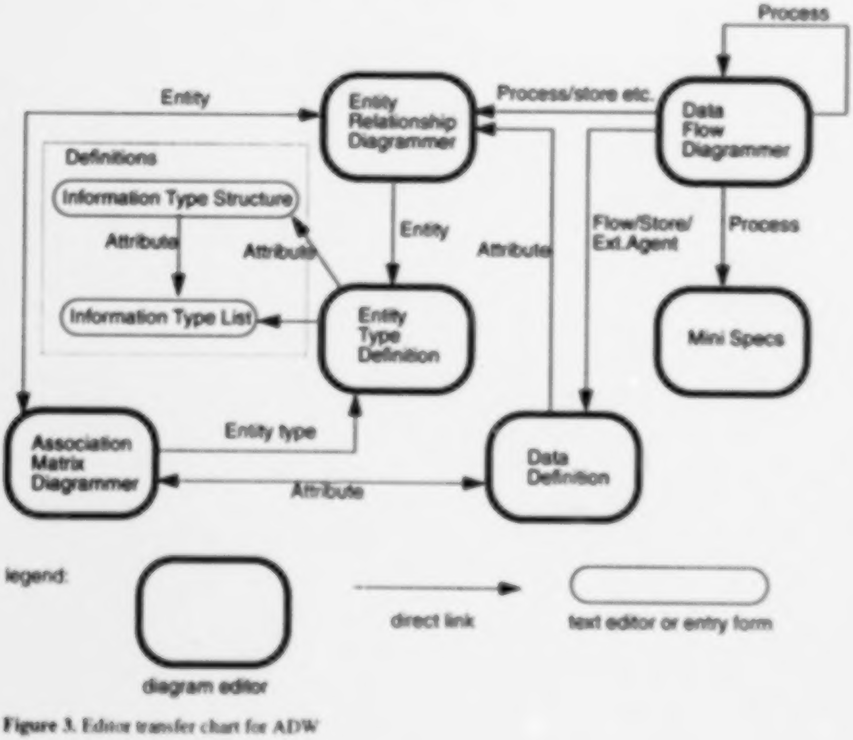

Process
Entity
Process/store etc.
Entity Relationship Diagrammer
Data Flow Diagrammer
Definitions
Information Type Structure
Attribute
Attribute
Entity
Attribute
Flow/Store/ Ext.Agent
Process
Information Type List
Entity Type Definition
Mini Specs
Association Matrix Diagrammer
Entity type
Data Definition
Attribute
legend:
direct link
text editor or entry form
diagram editor

Figure 3. Editor transfer chart for ADW

An editor transfer chart for ADW is shown in Figure 3. It shows that it is possible to transfer from the data flow diagram editor to the entity relationship diagram editor by clicking a process or a store in a DFD, but the reverse transfer is not supported by the corresponding CASE tool. This means that the data flow diagrammer can only be started by explicit commands issued by the CASE tool user. The association matrix diagrammer shows how components are mutually related; this kind of editor is present in ADW, Oracle*Case and FOUNDATION. From the matrix diagrammer it is possible to transfer to the entity type definition editor and the data definition editor. Decomposition in a DFD is represented by the loop with label *Process* from the data flow diagrammer block back to itself. Processes in DFDs can be specified in detail by the 'mini specs' editor.

4. Formalization of complexity reduction

In this section a brief formal discussion on conceptual complexity reduction techniques is presented.

Decomposition

Decomposition is treated in the most general case, viz. for diagrams to be represented as labelled graphs:

G is a labelled graph $G = \langle N, E \rangle$; (brackets denote tuples)
N is the set of nodes of the graph (the internal nodes): $N \neq \emptyset$;
M is the set of nodes in the environment of G (the external nodes): $N \cap M = \emptyset$
E is a set of labelled edges: $E \subseteq (N \cup M) \times (N \cup M) \times L$;
L is a set of labels; a label contains specific information of an edge and may differentiate between edges with common nodes in multigraphs;
an edge e may be denoted by a triple $\langle n_1, n_2, l \rangle$ with $n_1, n_2 \in (N \cup M)$ and $l \in L$;
$\forall \langle n_1, n_2, l \rangle \in E\ [n_1 \in N \vee n_2 \in N]$;

E_n is the set of edges incident into or from node n, $n \in N$:
$E_n = \{\langle n_1, n_2, l \rangle \in E \mid n_1, n_2 \in (N \cup M) \wedge (n_1 = n \vee n_2 = n)\}$;
D_n is the set of nodes directly connected to node n:
$D_n = \{p \in (N \cup M) \mid \exists\, e = \langle n_1, n_2, l \rangle \in E_n\ [n_1 = p \vee n_2 = p]\}$;

The decomposition of the graph $G = \langle N, E \rangle$ with respect to node $n \in N$ is a graph $G' = \langle N', E' \rangle$:
N' is a set of nodes: $N' \neq \emptyset$, $(N \cup M) \cap N' = \emptyset$;
M' is the set of nodes in the environment of G': $M' = (N \cup M)$
E' is a set of labelled edges: $E' \subseteq (N' \cup D_n) \times (N' \cup D_n) \times L$;
$(\forall \langle n_1, n_2, l \rangle \in E' \mid (n_1 \in D_n \wedge n_2 \in N') \vee (n_2 \in D_n \wedge n_1 \in N'))\ [\exists!\, e \in E_n]$;
$\forall e \in E_n\ [\exists!\, e' \in E']$.

The formalism can be applied recursively; the inclusion of the environment in the definition prevents loops from occurring. The formalism can be applied to DFDs; in this case the internal nodes have to be distinguished into processes and data stores; edges correspond to data flows. In the top level or the context diagram the environment contains the external agents. Decomposition can only be applied with respect to processes.

The preceding formalism does not prevent reuse, because the same node may occur in decompositions of different nodes. Suppose a graph G contains nodes n_1 and n_2; the decompositions of these nodes may both contain a node n_3, so these 'parallel' decompositions do not need to be disjunct, see also Huber et al. (1991).

Viewing, hiding

A view is derived from a total diagram by taking only a limited part of the whole. The total diagram is again represented by a labelled graph G:

$G = \langle N, E\rangle$;

N is the set of nodes of the graph: $N \neq \emptyset$;

E is a set of labelled edges: $E \subseteq N \times N \times L$;

a view of G is a graph $G' = \langle N', E'\rangle$:

$N' \subseteq N, N' \neq \emptyset$;

$E' \subseteq N' \times N' \times L \subseteq E$.

Hiding is conceptually the same as viewing, viz. omitting concepts from a more extensive diagram. The pragmatic difference between these techniques is that usually several views based on the same global diagram exist, whereas hiding provides just another version of a specific diagram with less detail.

5. Results of investigations

In this study eight commercial CASE tools were investigated:

- ADW (Application Development Workbench), version 2.7, manufactured by KnowledgeWare,
- Bachman Product Set, version 4.10, manufactured by Bachman Information Systems, Inc.,
- Excelerator II, version 1.2: Structured Method (SM), manufactured by Intersolv,
- Foundation: DESIGN/1 version 6.02, PLAN/1 version 1.1, manufactured by Andersen Consulting,
- Maestro II: SEtec version 1.15, manufactured by Softlab GmbH,
- Oracle*Case: CASE*Designer version 1.1, CASE*Dictionary version 5.0, CASE*Generator version 2.0, manufactured by Oracle,
- SDW (System Development Workbench) version 2.0, manufactured by Cap Volmac,
- System Architect, version 2.4G, manufactured by Popkin Software & Systems, Inc.

Complexity reduction in CASE tools

In an investigation the complexity reduction and the graphical techniques of the eight CASE tools mentioned above were assessed. The results are compiled in Table 1. Obviously, viewing, decomposition, zooming and scrolling are very general facilities that are supported in virtually all CASE tools. The features offered by the CASE tools investigated are adequate, though a larger set of facilities may be an indication for a higher level of technical sophistication.

Modelling transparency in CASE tools

From our investigation it appeared that in most CASE tools investigated only a subset of all possible dependencies have been implemented, which means that such tools do not have the full functionality of modelling transparency degree 2, so their transparency degree has a value somewhere between 1 and 2. Another conclusion is that in practice hypertext functionality is often available in addition to direct links, which means that manufacturers strive for more transparency by applying both direct links and hypertext functionality simultaneously. A classification of CASE tools in terms of their transparency degrees appears for this reason less relevant. Another way of assessing modelling transparency is by taking both direct links and hypertext functionality into account. This yields a two-dimensional diagram in which CASE tools can be located as shown in Figure 4.

On the vertical axis of the diagram support of direct links (transparency degree 2) is represented. The origin of the diagram corresponds to transparency degree 1: no support

Table 1. Complexity reduction and graphical techniques in CASE tools

	ADW	Bachman	Excelerator	FOUNDATION	Maestro II	Oracle*CASE	SDW	System Architect
Viewing	✗	✗	✗	✗	✗	✗	✗	✗
Decom-position	✗	✗	✗	✗	✗	✗	✗	✗
Zooming	✗	✗	✗	✗	✗	✗	partially	✗
Scrolling	✗	✗	✗	✗	✗	✗	✗	✗
Off-page connectors			✗	✗	✗	✗	✗	✗
Colouring	✗		✗	✗				
Automatic layout	✗	✗	✗			✗		✗
Hiding	✗	✗	✗			✗		✗
Adaptable symbols			✗		✗			✗

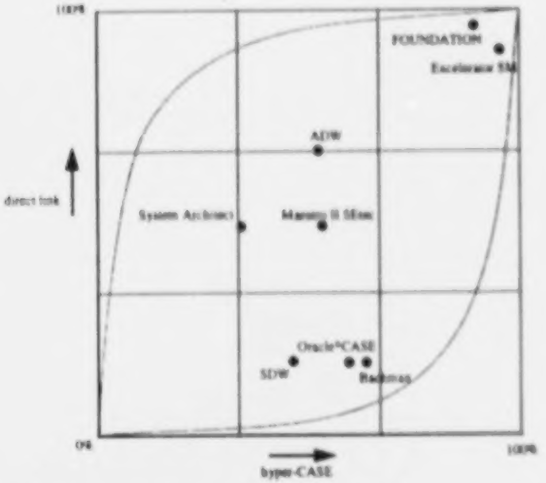


Figure 4. Direct link and hyper-CASE functionalities in CASE tools

of direct links; 100% on the vertical axis corresponds to full support of all relevant direct links. The ratio between actually supported direct links and full support is expressed by a percentage on the vertical axis.

The horizontal axis represents support of hyper-CASE functionality. The maximum of hyper-CASE corresponds to full user access to the meta-datamodel of the repository of the CASE tool. This corresponds to the possibility to define all possible dependencies or a completely accessible meta-model. In practice there is some dependency between the two axes. In case of full hyper-CASE functionality, usually also all direct links are supported, and vice versa. For this reason CASE-tools appear to be situated in a limited area in the diagram bounded by the two curves drawn in the diagram. Excelerator and FOUNDATION support hyper-CASE functionality to a high extent. Maestro II itself is a meta-CASE tool offering the user the possibility to define and generate his own CASE tools; SEtec is a CASE tool realized with Maestro II, which is marketed by the manufacturer.

6. CONCLUSIONS

The flexibility of CASE tools can be characterized by the facilities that they provide for complexity reduction and modelling transparency. The degree of modelling transparency in particular expresses the level of technical maturity of the tools as it is a good measure for the support of non-trivial user operations.

Eight commercially available CASE tool have been compared in terms of these features. It should be stipulated that the evaluation is rather time dependent; new and more powerful versions of the investigated and other CASE tools are becoming available. Transparency facilities are expected to be extended and improved considerably in the near future. Especially hyper-CASE functionality will be available in an increasing number of CASE-tools.

Acknowledgements

Acknowledgements are due to our colleague George Steenbekke, and to the students Learco Brizzi and Robin Wildschut, for their cooperation in this project. We gratefully acknowledge the facilities and support provided by the manufacturers and distributors of the CASE tools used in this investigation.

REFERENCES

S. Brinkkemper, (1993), "Integrating diagrams in CASE tools through modelling transparency", *Information and Software Technology*, Vol. 35, No. 2, February 1993, p. 101 - 105.

P. Coad and E. Yourdon, (1990), "Object-Oriented Analysis", Prentice Hall.

R. Elmasri and S. B. Navathe, (1994), "Fundamentals of Database Systems", 2nd edition, Benjamin/Cummings.

D.J. Hatley and I.A. Pirbhai, (1987), "Strategies for Real-Time System Specification", Dorset House Publishing Co.

P. Huber, K. Jensen and R.M. Shapiro, (1991), "Hierarchies in coloured Petri nets", *in*: G. Rozenberg (Ed.), Advances in Petri Nets 1990, Lecture Notes in Computer Science, 483, Springer-Verlag.

C.A.M. Koesen, S. Brinkkemper and H.E. Keus, (1989), "The layered modelling of dialogues and its support workbench", *in*: J. Jenkins (Ed.), "Advance working papers, Third Int. Conf. on Computer Aided Software Engineering", Imperial College, London, July 1989, p. 87 - 107.

J. Martin and C. McClure, (1988), "Structured Techniques: the Basis for CASE", Prentice Hall.

H. L. Ossher, (1987), "A mechanism for specifying the structure of large, layered systems", *in*: B. Shriver and P. Wegner, (Eds.), "Research Directions in Object-Oriented Programming", MIT Press, p. 219 - 252.

M.F. Rothstein, B. Rosner, M.A. Senatore and D. Mulligan, (1993), "Structured analysis and design for the CASE user", McGraw-Hill.

E. Yourdon, (1989), "Modern Structured Analysis", Prentice Hall.

THE USE OF OBJECT MODELS FOR INFORMATION SYSTEM ANALYSIS

Ying Liang[1], Mike Newton[2] and Hugh Robinson[2]

[1]University of Paisley, Department of Computing and Information Systems, Paisley PA1 2BE, U.K.
[2]The Open University, Department of Computing, Milton Keynes MK7 6AA, U.K.

Abstract

Three aspects—data, process, behaviour aspects—of information system requirements are fundamental concerns in analysis. Thus models provided for information system specification need to consider and represent one or more of them. Object models are now being promoted as another kind of model for information system specification. It is significant to find out how the three basic aspects of information system requirements are represented by object models. Four object-oriented methods were used to analyse a book trader system. The results of the modelling are shown in this paper. On the basis of the results, this paper compares the four object models in the methods by addressing the object abstractions included in the models, in particular, their concerns for the three basic aspects of information system requirements, and their significance for the identification and specification of objects in analysis.

1. INTRODUCTION

For information system specification, the issue of the aspects of information system requirements that require abstraction and specification is as significant as the issue of the techniques used to represent them. The nature of these aspects decides the things that a software system should do in accordance with information system requirements. Therefore, a system development method always provides a model for information system specification. The information system specification is represented by the model. On the other hand, in terms of models, an analyst can understand what objective or quantified information he/she should identify and specify from information system requirements.

Generally, three aspects, i.e. data, process, and behaviour, of information system requirements are regarded as fundamental aspects of concern in analysis, and they are usually emphasised and represented in terms of models provided by the methods (See Figure 1). The degree to which any given method emphasises each of the three aspects varies. For example, the modelling in structured analysis (Yourdon (1989)) emphasises the process aspect, the modelling in entity-relationship modelling (Chen (1980)) focuses on the data aspect and the modelling in JSP (Jackson (1983)) stresses the behaviour aspect; whereas SSADM seeks to emphasise all three aspects within its framework (Downs

(1992)). The emphasised aspect of information system requirements in a model has strong impact on the discovery and invention of the essential components (such as entity or module, etc.) in the information system specification.

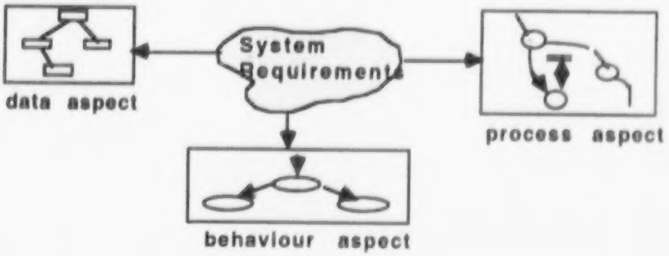


Figure 1. Three basic aspects of information system requirements

Object models are now being promoted for the information system specification. In order to understand object-oriented analysis and models, we carried out a study (Liang, Newton and Robinson (1993)) in which four representative object-oriented methods (Coad and Yourdon (1991), Rumbaugh (1991), Booch (1991), Wirfs-Brock *et al* (1990)) were used to analyse a book trader system. The study illustrates that one or more basic aspects of information system requirements are still addressed by the methods and they are reflected in the object models offered by the methods. These object models emphasise one or more of the three basic aspects of information system requirements underlying the definition of the concept of 'object' in each method.

2. FOUR OBJECT MODELS AND THEIR APPLICATIONS TO INFORMATION SYSTEM SPECIFICATION

In the study (Liang, Newton and Robinson (1993)), the object models offered by the four object-oriented methods were used to describe the requirements of a book trader system through following the processes of analysis prescribed by the methods. Each of these methods was applied to the same problem scenario: modelling the information processing requirements of a company whose business is the selling of books by advertising in magazines. Customers can buy books from the company, at the price stated in an advertisement, by mail order (with payment by cheque or credit card) or on the telephone (payment by credit card only). The outcomes of the modelling process for the four methods are now discussed.

2.1 The Object Model in the OOA Method

A five-layer object model is provided by the OOA method (Coad and Yourdon (1991)), which is used to describe objects, classes, their attributes, their structures, and their operations performed on the attributes. In our study, such an object model was generated for a book trader system by using the method. It is shown in Figure 2 and it specifies the requirements of the book trader system. The description of the states in the object 'Order' is given in Figure 3 which is connected with the service layer in Figure 2. The study shows that the OOA method focuses on the data aspect of the problem since the operations within objects and classes are defined as the actions on the attributes that are the persistent data in the model. In addition, the states in each object are also considered by the method and they are individually specified in the model.

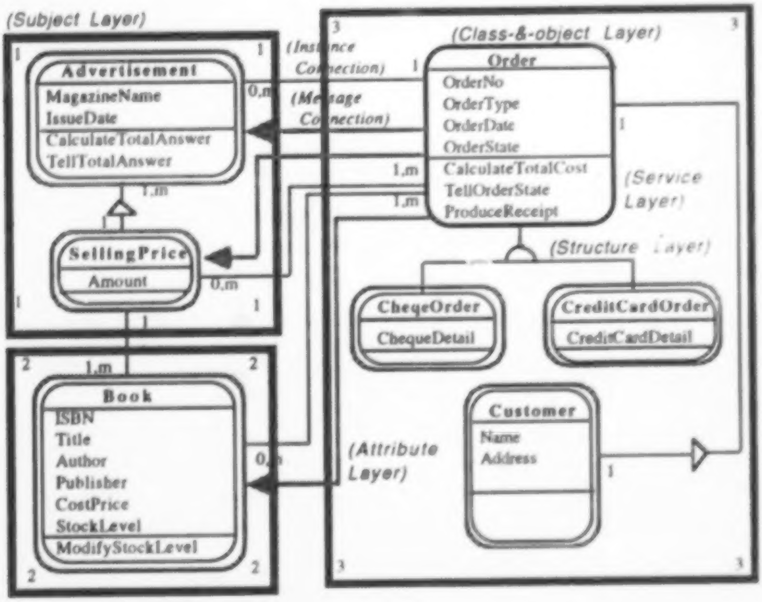

(Subject Layer)
(Class-&-object Layer)
(Instance Connection)
(Message Connection)
(Service Layer)
(Structure Layer)
(Attribute Layer)
Advertisement
MagazineName
IssueDate
CalculateTotalAnswer
TellTotalAnswer
SellingPrice
Amount
Book
ISBN
Title
Author
Publisher
CostPrice
StockLevel
ModifyStockLevel
Order
OrderNo
OrderType
OrderDate
OrderState
CalculateTotalCost
TellOrderState
ProduceReceipt
ChequeOrder
ChequeDetail
CreditCardOrder
CreditCardDetail
Customer
Name
Address
1
0,m
1,m
2
3

Figure 2. An object model of a book trader system in OOA method

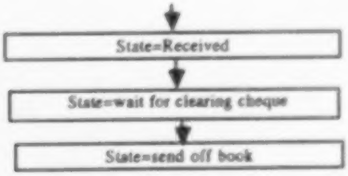

State=Received
State=wait for clearing cheque
State=send off book

Figure 3. An object state diagram of the object "ChequeOrder"

2.2 The Object Model in the OMT Method

An object model is produced in the OMT method (Rumbaugh (1991)) to model information system requirements. This model is concerned with three basic aspects of information system requirements: data aspect—objects, classes, their persistent data, and their relationships; behaviour aspect—events, states of objects and classes, and interactions between objects; and process aspect—inputs, outputs, processes within objects and classes. Such an object model was built in the case study when using the OMT method to analyse the book trader problem. This model is shown in Figure 4, 5 and 6 each of which illustrates a specific model corresponding to each aspect concerned. The study shows that the OMT method emphasises both the data aspect and the process aspect in analysis. The data aspect of information system requirements, i.e. attributes, is considered and defined first in the object model by the OMT method. Then the behaviour of objects is defined in terms of the attributes defined. The process aspect can be either specified in terms of the attributes in the object model or the actions in the dynamic model.

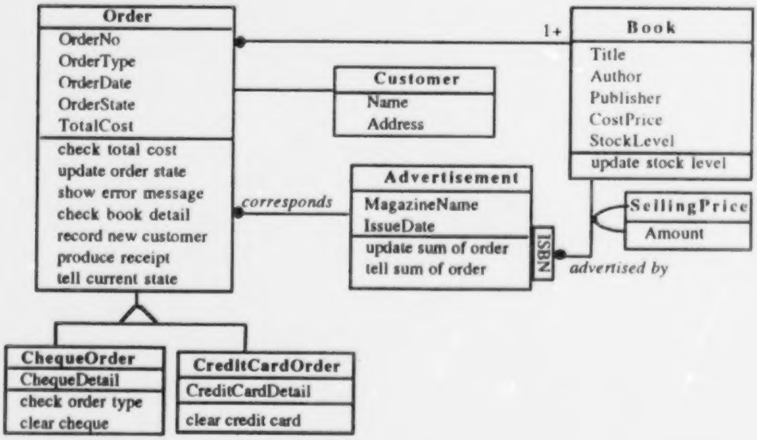

Order
OrderNo
OrderType
OrderDate
OrderState
TotalCost
check total cost
update order state
show error message
check book detail
record new customer
produce receipt
tell current state
1+
Book
Title
Author
Publisher
CostPrice
StockLevel
update stock level
Customer
Name
Address
corresponds
Advertisement
MagazineName
IssueDate
update sum of order
tell sum of order
ISBN
advertised by
SellingPrice
Amount
ChequeOrder
ChequeDetail
check order type
clear cheque
CreditCardOrder
CreditCardDetail
clear credit card

Figure 4. A model of a book trader system—data concerned

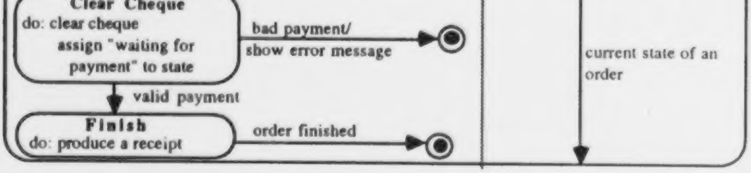

ChequeOrder
order arrives
Initial
do: assign "received" to state
record new customer
check book details
bad details/
show error message
valid details [OrderType is mail]/check total cost
Clear Cheque
do: clear cheque
assign "waiting for payment" to state
bad payment/
show error message
valid payment
Finish
do: produce a receipt
order finished
enquiry an order
[in Clear Payment]
Order Enquiry
do: tell current state
of the order
current state of an
order

Figure 5. A model of a book trader system—behaviour concerned

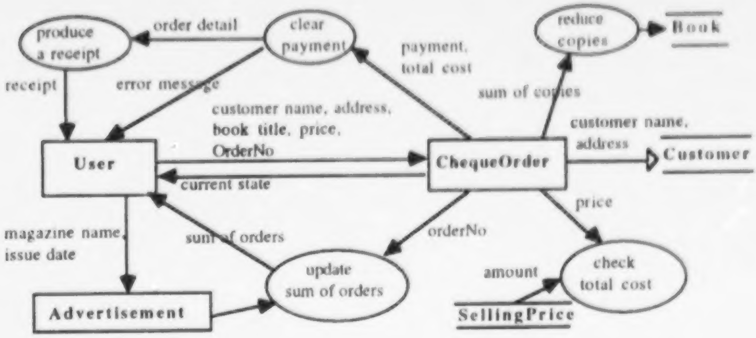

produce a receipt
order detail
clear payment
payment, total cost
reduce copies
Book
receipt
error message
sum of copies
customer name, address, book title, price, OrderNo
customer name, address
User
ChequeOrder
Customer
current state
price
magazine name, issue date
sum of orders
orderNo
update sum of orders
amount
check total cost
Advertisement
SellingPrice

Figure 6. A model of a book trader system—process concerned

2.3 The Object Model in Booch's Method

An object model is provided and used to describe information system requirements by Booch's method (Booch (1991)). In terms of this model, objects and classes, relationships between them, operations within them, and the behaviour of them can be defined. In the case study, such an object model was constructed when applying Booch's method to describe the book trader problem. This model is shown by Figure 7, 8 and 9. In the model, objects and classes are specified separately by object diagrams (Figure 7) and class diagrams (Figure 8). An object diagram describes the interactions between objects; while a class diagram specifies the operations and behaviour within each class (Figure 9), and the relationships between classes. The study illustrates that Booch's method emphasises process aspect of information system requirements. The states and data within objects and classes are defined in terms of the processes that must be fulfilled by the objects and classes.

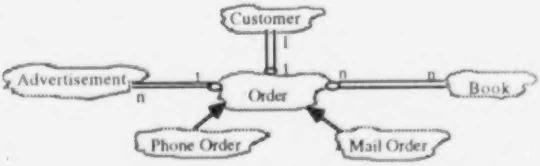

Customer
1
1
Advertisement
1
n
Order
n
n
Book
Phone Order
Mail Order

Figure 7. Class description in the object model of a book trader system

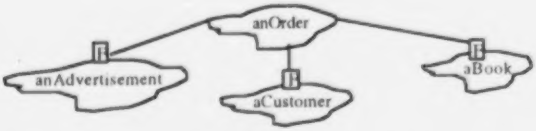

anOrder
anAdvertisement
aCustomer
aBook

Figure 8. Object description in the object model of a book trader system

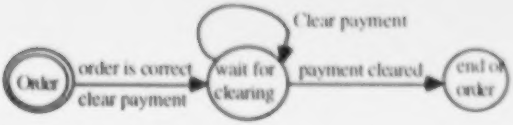


Figure 9. Behaviour description of class 'Order' in the object model

2.4 The Object Model in Wirfs-Brock's Method

The object modelling supported by Wirfs-Brock's method (Wirfs-Brock *et al* (1990)) permits the specification of classes, operations within them, and hierarchies and collaborations between them. The model for our study generated by the modelling process is shown in Figure 10. This diagram is a collaboration graph that shows the objects and classes and their collaborations. Then each class will be described in terms of a class card that shows the operations within the class. It also shows the client/server relationships (i.e. contracts) between its operations and the operations within other classes. This study shows that Wirfs-Brock's method focuses on the process aspect of information system requirements in system analysis. The data within classes is not specified explicitly, instead it is implicitly defined in the operations that must be fulfilled by the classes.

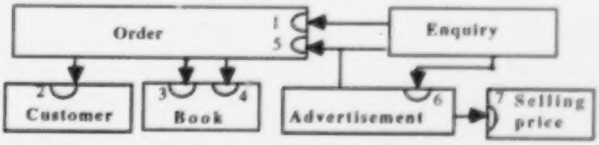


Figure 10. Class description in the object model of a book trader system

3. OBJECT ABSTRACTIONS IN THE OBJECT MODELS

In general, an object can be regarded as an encapsulation of data and operations. The central problem for an object-oriented method is that of identifying and assigning the appropriate semantics (i.e. data, operations, or states) to objects that adequately model the problem. The experience of applying the four object-oriented methods show that the object models in the methods are commonly built by first identifying the objects from information system requirements and secondly identifying the appropriate semantics of the objects and assigning them appropriately. Such semantics are identified and specified by considering the basic aspects of the information system requirements in accordance with the objects. Based on the results of the study, the object abstractions in the four object models in the methods can be illustrated by a diagram as shown in Figure 11. This diagram shows that the object abstractions in the four object models are always concerned with one or more basic aspects of the requirement. The three aspects have impacts on the identification and definition of the semantics of the objects in the information system specification. By using this diagram, we can easily show the object abstractions in an object model by looking at the concerns included in the abstractions, and understand the ways in which data and processes are encapsulated into objects.

Underlying the concerns of objects abstractions in the object models, an object-oriented method provides a modelling process to identify and define the objects from information system requirements. The study shows that the four object-oriented methods provide different ways to do analysis because they have different concerns for object abstractions in their object models. The relationships between the concerns of the object abstraction in the object models and the processes of analysis in the four object-oriented methods are given in Figure 12. It shows that an object-oriented method may identify and define objects by addressing different concerns of the abstractions in analysis.

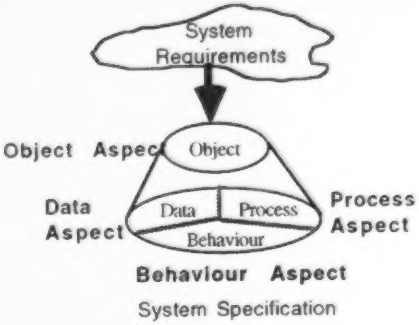

System Requirements
Object Aspect
Object
Data Aspect
Data
Process
Process Aspect
Behaviour
Behaviour Aspect
System Specification

Figure 11. Object abstractions in the object models

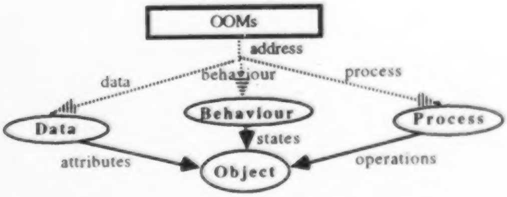

OOMs
address
data
behaviour
process
Data
Behaviour
Process
states
attributes
operations
Object

Figure 12. The concerns of object abstractions and the process of analysis

4. CONCERNS OF OBJECT ABSTRACTIONS IN THE OBJECT MODELS

By looking at the object models shown in previous section, we can see that the four models have different concerns about the basic aspects of information system requirements in their object abstractions. In particular, the four object-oriented methods emphasise some aspects by providing a specific process of analysis to enforce analysts to consider and specify the aspects explicitly in the object models.

4.1 Concerns of the Object Abstractions in the OOA Object Model

The results of the study illustrate that the OOA method is concerned with data and behaviour aspects and includes them in the object abstractions in the OOA object model. The data aspect is specially emphasised by the method: the persistent data are specified as the attributes within objects and the relationships of data are defined as the association relationships between the objects. The operations within objects are defined in terms of the attributes within the objects and they will be performed on these attributes. So the process aspect is implicitly considered by the OOA method. The behaviour aspect is represented as the states of objects by considering the changes of the attribute values within the objects.

4.2 Concerns of the Object Abstractions in the OMT Object Model

The results of the study demonstrate that OMT method is concerned with all three basic aspects of information system requirements, in the object abstractions of the OMT object model. The persistent data in information system requirements is defined as the attributes within objects and the relationships of data are specified by link relationships between objects. In addition, the processes for data transformations in the requirements are represented as the operations within objects. The behaviour aspect is specified as the behaviour of individual objects in terms of the attributes and the processes within the object. Thus the data aspect is strongly emphasised by the OMT method.

4.3 Concerns of the Object Abstractions in Booch's Object Model

The results of the study show that the concerns of the object abstractions in Booch's object model include the process and behaviour aspects of information system requirements. The process aspect is specified as the operations within objects. The behaviour aspect is specified as the behaviour of each object in terms of the operations that the object needs to fulfil. The persistent data is not specified as the attributes within objects. Instead it is defined as the parameters in the operations within objects. The communication of data is implied in the interactions between objects through message passing. So the data aspect is indirectly referred to when specifying the protocols of the operations within objects.

4.4 Concerns of the Object Abstractions in Wirfs-Brock's Object Model

The results of the study illustrate that object abstractions in Wirfs-Brock's model are only concerned with the process aspect of information system requirements. This aspect is specified as the responsibilities (also operations) that objects should perform. The cooperations between processes are specified as the collaborations between objects. Like Booch's object model, the data aspect is not specified in particular in Wirfs-Brock's model. Instead the persistent data is defined as the parameters in the responsibilities. The communication of data is implied in the collaborations between objects. So the data aspect can be considered to be implicitly specified by Wirfs-Brock's method.

In this section, the concerns of the object abstractions in the four object models in the methods are discussed. As a conclusion, they are summarised by drawing the diagrams as shown in Figure 13. In the diagrams, different toned parts are used to express the varied degree of the concerns of the object abstractions in each model: white parts represent the emphasised concerns in an object model; *light grey* parts and *dark grey* parts mean the explicit concerns or implicit concerns of the object abstractions in the four object models; *black* parts express the aspect which is not considered by the method. Underlying the concerns of the object abstractions, the ways to identify and define the objects for information system specification are decided and provided by the four object-oriented methods. They are shown in Figure 14.

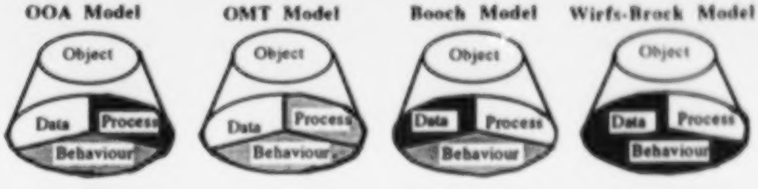

OOA Model
OMT Model
Booch Model
Wirfs-Brock Model
Object
Data
Process
Behaviour

Figure 13. Concerns of the object abstractions in each object model

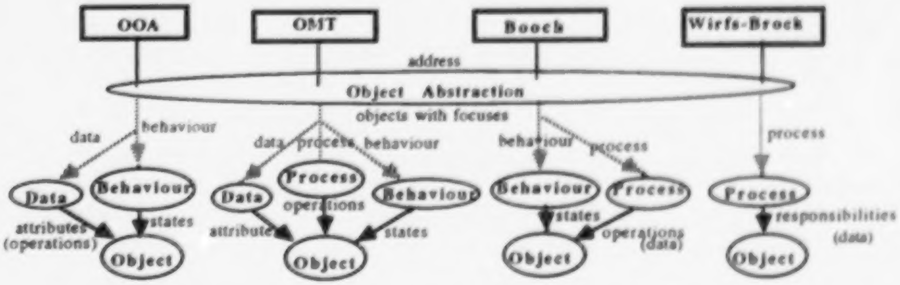

OOA
OMT
Booch
Wirfs-Brock
address
Object Abstraction
objects with focuses
data
behaviour
process
Data
Behaviour
Process
attributes (operations)
states
attributes
operations
operations (data)
responsibilities (data)
Object

Figure 14. The ways to the object abstractions in the object models

5. THE USE OF THE OBJECT MODELS

So far we have discussed the concerns of the object abstractions in the object models in four object-oriented methods, based on the object models of a book trader system that were produced by a study (Liang, Newton and Robinson (1993)). The four methods carry out object abstraction with different concerns on the three basic aspects of information system requirements. Such differences not only lead to different ways to do analysis, but also, more importantly, produce the different object models for information system specification, like the ones shown in Figure 2—10. This study shows the case in which the concerns of an object abstraction in object models have strong impacts on the use of the object models, for information system specifications in a wide range.

For information processing problems, the data and the operations on the data are usually thought important and significant to identify and specify in the specifications of information systems. In this case, an object-oriented method that emphasises data aspect of information system requirements may be chosen, since the object abstractions in its object model are concerned with the data aspect and the attributes within objects can be defined explicitly. In particular, the persistent data which are shared by the objects can be also considered and specified by the model. On the other hand, for scientific calculation problems, the processes that transform inputs to outputs are commonly considered important and significant to identify and specify in the specifications of scientific calculation systems. The object-oriented method that focuses on process aspect of information system requirements can be chosen, since the object abstractions in its object model will specially support the identifications and definitions of the operations within objects. In contrast to the object model that emphasises data aspect, the inheritance relationships between objects and classes are often specified through considering the common processes within the objects.

However for procedure control problems, instead of data and processes, the events that decide the changes of states in the procedure are usually considered important and significant to identify and specify in the specifications of procedure control systems. The object-oriented method that stresses behaviour aspect of information system requirements may be chosen to use as the object abstractions in its object model should define the behaviour in particular. The inheritance relationships between objects and classes are often specified through considering the common behaviour of the objects. For example, in our study (Liang, Newton and Robinson (1993)), since the book trader problem is an information processing problem, by considering the essential features of an information system, the OOA and the OMT object models may represent the book trader problem more precisely and completely than Booch's and Wirfs-Brock's object models, since they emphasise data aspect of the requirements in their object abstractions, and they provide the specific notations to specify the data aspect in separation with other aspects.

An object model usually emphasises one basic aspect of information system requirements, that is, the model often is not suitable to specify some kind of systems in which other aspect is more important. Thus the efficiency and completeness of the information system specification should be considered carefully by an analyst. For instance, Booch's and Wirfs-Brock's object models emphasise the process aspect but data aspect. However they still can be used to specify information systems, as we did in our study (Liang, Newton and Robinson (1993)). However the experience of the study tells that Booch's and Wirfs-Brock's object models are not appropriate to use to specify a book trader system since they do not emphasise the data aspect and they do not provide the significant notations with the guidelines and criteria to help us to specify the data aspect that is essential to the problem. In addition, the information system specification of the book trader system that specified by the OOA and the OMT object models is more understandable, since the persistent data is defined explicitly in terms of attributes of objects, and the operations are specified by referring to the data.

6. REFERENCES

Booch, G., (1991), "Object-Oriented Design with Applications", The Benjamin Cummings Publishing Company, CA.

Chen, P., (1980), "Entity-Relationship Approach to System Analysis and Design", North-Hollan, Amsterdam.

Coad, P. and Yourdon, E., (1991), "Object-Oriented Analysis", Prentice Hall, New Jersey.

Downs E., Clare P., and Coe I., (1992), "Structured Systems Analysis and Design Method", Prentice Hall, New Jersey.

Jackson, M.A., (1983), "Principles of Programming Design", Academic Press.

Liang, Y., Newton, M.A., and Robinson, H.M., (1993), Analysis of information systems using object-oriented methodologies, *Proceedings of BCS ISM SG & BSS Joint Conference on the Theory, Use and Integrative Aspects of IS Methodologies*, Edinburgh, pp.57-70.

Rumbaugh, J. et al, (1991), "Object-Oriented Modelling and Design", Prentice Hall, New Jersey.

Wirfs-Brock, R.J. et al, (1990), "Designing Object-Oriented Software", Prentice Hall, New Jersey.

Yourdon, E., (1989), "Modern Structured Analysis", Prentice Hall, New Jersey.

INTEGRATING EXPERT SYSTEMS APPLICATIONS INTO MAINSTREAM BUSINESS OPERATIONS.

Andrew Shie [1] & Robert Moreton [2]

[1] Bolton Business School, Bolton Institute, Deane Road, Bolton BL3 5AB, England
[2] School of Computing & Information Technology, University of Wolverhampton, Wulfruna Street,
Wolverhampton WV1 1SB, England

Abstract

Expert systems has proved to be a useful and feasible technology. A recent report estimated that in Japan, there are about 1,000 to 2,000 fielded KBS, while in the U.S., the number is "probably several times that of Japan" (JTEC,94). Expert systems offer an opportunity to formalise existing knowledge and expertise for dissemination within the organisation, thereby adding value to the business. Recently available methodologies e.g. STAGES, KADS are also encouraging interest in the technology. However, these have mainly focused on stand-alone developments. Integration into mainstream business operations is necessary if expert or knowledge-based systems technology is to realise its potential. As with conventional applications, they have to be planned for. Through strategic planning, knowledge-based systems can be focused in areas where knowledge and expertise is much needed i.e. be relevant to the business, and have an impact on the overall performance of the business.

The paper will present the work from on-going research to develop a business-centred approach for knowledge-based systems development. It begins by describing a framework for assessing the adequacy of development methodologies. The results from evaluating three development methodologies: SSADM, IEM and KADS, are then presented. This is followed by some proposals for improving one particular methodology to highlight knowledge and expertise pockets during its planning stage. The paper concludes with a description of some current work with an engineering company in which the extended methodology is being applied.

1. INTRODUCTION.

Information is no longer the distinguishing mark of a successful organisation. Knowledge or knowing what to do with information is (Rock,93). It has also been suggested that knowledge will be "the key strategic resource by the 21st century"

(Melody,90). Expert or Knowledge-based systems (KBS) technology offers an opportunity to manage these valuable assets (Shie,93). The technology offers the ability to formalise, preserve, duplicate and disseminate existing knowledge and expertise. In addition, they could also be used to increase the availability of knowledge within the organisation, facilitating opportunities to gain "lasting competitive advantage" (Nonaka,91).

However there has only been limited success in using and deploying KBS. A survey published in 1991 (DTI,91) estimates that in the UK, there are only 700 user-companies, with a projected world-wide usage of 2,000. Of the 700, the survey revealed that there were only 199 active users. There are a number of possible reasons for this which include: the lack of adequate development methods and techniques (the same survey estimates that about 45%-68% of the KBS developed were based on prototyping or incremental developments), development difficulties e.g. knowledge bottleneck (Liou,92), difficulties in identifying potentially useful applications, the need to provide business advantages, the need for integration with existing systems (Khalr,91) etc.

There have been a number of initiatives which have emerged to counter these problems: the ESPRIT initiative, which produced the KADS methodology (Bright et al,91); the GEMINI initiative (Scarff,91); the compilation of success and failure cases into a knowledge-base, for guiding future KBS developments (Journal,92) etc. They are encouraging interest in the technology through the availability of new methodologies, methods and techniques.

This research paper is concerned with KADS, in particular its adequacy as a development methodology. Whilst there has been a significant body of work defining and evaluating conventional systems development methodologies, there has been relatively few attempts to deal with KBS methodologies e.g. Hilal & Soltan(91) and BAe(90). Typically these have been concerned with evaluating one KBS methodology against another.

The KADS methodology has been suggested as the emerging de facto standard for KBS developments in Europe (Hickman et al,89) and is gaining interest in the U.S. It also forms the foundation for the second ESPRIT KADS-II initiative to develop the Common KADS method (KADS,92), which is currently underway.

2. AN EVALUATIVE FRAMEWORK FOR ASSESSING DEVELOPMENT METHODOLOGIES.

An evaluative framework was developed by Shie & Sumiga(92) which contains twelve elements for consideration in assessing the adequacy of a development methodology. The twelve elements are: Orientation, Perspective, Objective, Techniques and Tools, Strategic Outlook, Life-cycle model, Range, Usability, User-involvement, Future changes, Integration and Evaluation.

A methodology represents "a method of developing systems, with identified phases and sub-phases, recommended techniques to use in each phase and sub-phase, and recommendations about planning, management, control and evaluation" (Lawson,90). There is an underlying philosophy, which determines the focus of attention, the manner for development and the final end-product(s). It also determines the accompanying techniques and tools. The framework assesses this through the first three elements:

Orientation, Perspective and Objective. Orientation highlights the main focus of attention during the development. Currently, there are four: data, process, people and knowledge. Perspective determines the approach for the development activity. There are currently two main perspectives being used: systems and scientific. Objective examines the width of the solution space considered by the methodology during the development for instance, is a 'technical fix' always the solution or does it consider non-technical solutions to the problem.

The accompanying Techniques and Tools is an important consideration. Holloway describes a methodology as a "good kit of tools, where it provide the right tools to enable each developmental task to be successfully completed" (Holloway,89).

Whilst the Objective determines the width, Strategic Outlook examines the support provided by the methodology to identify potential application(s). This is usually through the incorporation of some form of planning activity in the initial stages of the development. It also assists in selecting the right techniques and/or tools to use in the development (Maddison,83).

An associated issue is the Life-cycle Model of the methodology. This represents the template of developmental activities covered by the methodology. Ideally, it "should cover the entire developmental life-cycle" (Wasserman, 83). There are also a number of models available: waterfall, Spiral (Boehm,88), Helix (Basden,89) etc. The concern is to assess the coverage of the development provided for by the methodology.

The Range should also be identified. This represents the generality of the methodology i.e. how applicable is the methodology for developing different classes or types of systems. Certain methodologies e.g. COMPACT claims to be suitable for developing certain classes of systems whilst others claim to be suitable for most classes of systems e.g. LBMS-SDM (Fitzgerald et al,85).

Usability is also important. This is "perhaps the most important characteristic of any methodology" (Thomas,91). It must also be "usable as well as adaptable" (Levine,85). However, this is difficult to assess, except through experience. In practice, the typical training requirements are highlighted, providing a 'feel' of its usability.

The amount of User-involvement, advocated by the methodology should also be considered. This is critical to the success of the development, as echoed in a study which indicated that "a high level of responsible participation, where appropriate, is a positive ingredient of successful systems development" (Avison & Wood-Harper,91). The task is to identify the various forms of user-involvement emphasised by the methodology.

Future Changes or extensibility of the methodology is another issue for consideration. Thomas(91) states that "a good expert systems methodology should be extensible over time". The approach(es) employed by the methodology to address this, are highlighted. A related issue is Integration. There is an increasing trend towards integrated systems. At this point of the evaluation, the question: 'Does it support integration ?' is asked.

The final consideration, Evaluation considers the support provided by the methodology, for evaluating end-products during the development. It is recognised that errors made at the earlier stages are relatively more expensive that those at later stages of the development (Ince,88). Furthermore, the cost of corrective and/or enhancement maintenance of software is estimated to be about 70% of the total cost (Aktas,87). The support in terms of the various methods of evaluation provided, are highlighted.

3. THE EVALUATION STUDY.

Although the main aim of the study was to evaluate KADS as a methodology, the research also included two other methodologies as part of the study: SSADM and IEM. They represent two examples of the current standards in terms of 'conventional' systems development methodologies and as such, provide the bench-mark of systems development methodological adequacy. SSADM was chosen because it has been the British civil service standard for conventional systems development since 1984 (Ashworth,88) while IEM claims to be the world's most widely used methodology (Whybrow,92). Furthermore, if KADS was found to be inadequate, these alternatives would form the basis for the enhancement(s). The results of the studies are summarised in the following table from Shie & Sumiga(92).

Table : Tabulated results of evaluating SSADM, IEM and KADS

ELEMENTS	SSADM	IEM	KADS
Orientation	Data	Data	Knowledge
Perspective	Scientific	Scientific	Scientific
Objective	Design specs	Info systems plan Design specs Encyclopaedia	Design specs
Techniques & Tools	Data-modelling CASE tool support	Data-modelling CASE tool support Use of 4GLs	Library of models CASE tool support
Strategic Outlook	None	Yes - ISP stage	None
Life-cycle model	Limited Analysis & Design	Full	Limited Analysis & Design
Range	DP systems in well-structured env.	DP systems in data-sharing env.	KBS
Usability	Not easily used	Skilled personnel	Proper training
User-Involvement	Reviews & sign-offs	User-driven	during Analysis
Future Changes	Stable data-model	Stable data model Stability analysis	Never-ending
Integration	None	Yes	None
Evaluation	User sign-offs	CSF analysis	Milestones, Workpackages, Deliverables

This analysis revealed that as a development methodology, KADS is limited in at least four aspects:

1. It does not consider the strategic implications of the development.
2. There is limited user-involvement during the development.
3. It does not address the issue of Integration.
4. The possible future business changes to the system are not considered.

In addition, IEM was found to be more adequate than the other two in the following areas: Strategic Outlook, Life-cycle Model, Future changes and Integration. The study also revealed the following series of good practices and methods:

SSADM	the cross-referencing of its 3 main models, for consistency and accuracy, emphasis of user reviews and sign-offs, before progression.
IEM	strategic planning as the initial stage of development, employing stability analysis on the data-model, the development of an encyclopaedia (Macdonald,86)
KADS	use of a library of interpretative models i.e. generic expertise models, parallel streams of consideration at analysis, ability to accommodate a shift in solution (Hickman et al, 89), attention to the modality issue (Hesketh & Barrett,90)

4. STRATEGIC PLANNING FOR KBS.

Although KADS represents one of the more established KBS methodologies, it is not as adequate as its conventional counterparts. There is also an increasing trend towards integrating conventional systems with KBS, which it does not address. KBS must also be relevant to the overall business strategy and benefit the organisation (Thomas & Adhami,91). As with conventional systems, they need to be planned in order for the organisation to reap the full benefit. Typically, this is done at a strategic level, through adopting an Information Strategy Planning (ISP) activity.

ISP represents an opportunity to plan and co-ordinate the development of the necessary information systems to support the organisation in achieving its overall business objectives. In the past, many systems were developed to meet certain local needs, leading to integration and compatibility problems. In addition, there was an increasing realisation that such systems has to be planned for, at a strategic level of the organisation so that they will feed the business operation of the organisation. Planning at such a level "aims to ensure that management needs are met" (Avison & Fitzgerald,88). ISP ensures that "the business procedures, information technologies and their associated costs are aligned with a company's strategic direction" (Battaglia,91). It also offers an opportunity to use information technology to create "competitive advantage" (Waema & Walshman,90).

There are various approaches for strategically planning the information needs of an organisation. They include: Earls' three-pronged methodology, IBM's Business Systems Planning, Norton's Investment Strategy Analysis, Martin's Information Engineering Methodology, Rockart's Critical Success Factors etc. Sprague & McNurlin(86) describes some of these in more detail. Of these, the one of particular interest to the research is Martin's Information Engineering Methodology. Unlike some, it represents a 'hard systems' approach to ISP. In addition, it provides the techniques and tools for developing an information strategy plan, various architecture and the production of required systems.

Some of its major benefits include: improved quality systems, improved productivity during development (Macdonald,86), real business systems, active user-involvement and control (Richmond,92), consistent and integrated standards, conceptual and structural strengths (Hares,92) etc. Furthermore, the research revealed that IEM is quite adequate as a development methodology (Shie & Sumiga,92).

5. INFORMATION ENGINEERING METHODOLOGY.

Developed from Martin's concept of Information Engineering which is described broadly as "data that is stored and maintained by computers and the information distilled from data" (Martin,82), IEM represents a seven-staged framework for "developing high-quality, integrated information systems by providing a basis for defining and managing projects (Macdonald,86). It also encourage the extensive use of CASE tools.

IEM begins with Information Strategy Planning, which develops an information strategy plan, containing three architectures: information, business systems and technical. In addition, it produces a prioritised list of systems required to support the organisation in achieving its business objective. Whilst ISP takes a top-level broad view of the organisation's needs, IEM's second stage, Business Area Analysis (BAA) focuses the development at a selected business area, to develop a business area description. This drives the third stage of development, Business Systems Design to develop an appropriate logical systems design, followed by a computer-based systems design at the stage of Technical Design. The development follows through the stages of Construction, Transition and Production. Martin(86) contains a fuller description of the methodology.

6. SOME POSSIBLE EXTENSIONS TO IEM.

However, like most ISP approaches, IEM was developed to "control and plan for information systems development" (Shie & Sumiga,92). Its scope is limited to information needs of the organisation. Whilst it is adequate at highlighting information needs, IEM does not assist in highlighting knowledge or expertise in the organisation. The research attempt to broaden its scope by suggesting the following extensions at the ISP stage:

1. the identification of knowledge elements.

2. the development of a knowledge model to enrich the information architecture.

The ISP stage begins by eliciting various business documents e.g. business plans, mission statements and transforming these into "a series of clearly defined sets of goals and objectives"(Davids,92). The goals and objectives determine the organisation's Critical Success Factors (CSFs). Performance measures of these goals and problems are also identified. Other tasks include the analysis of the business activities, the assessment of its current technical environment etc. As a result, an information strategy plan is produced. This contains three architectures: information, business systems and technical. The information architecture describe the information needs of the organisation's activities. Whilst the business systems architecture describe "naturally clustered business areas" i.e. natural groupings of business activities as derived from extensive analysis of its operations, the technical architecture describes the technical environment required to support the business systems architecture.

The first proposal involves the extension of two techniques, the Analysis of Problems and Goals, and the Analysis of CSFs. The first technique, the Analysis of Problems and Goals is used to identify and record individual departmental manager's problem(s) in achieving their goal, in relation to the organisation's business objective. Bottleneck(s) in achieving their goals are highlighted to be incorporated during the further analysis of that particular business area. The research propose to extend this to include the identification of any expert(s) and their role in the department, specifically their role in the bottle-neck. The second technique is the Analysis of CSFs, where the aim is to describe "the essential conditions" for the business to achieve its objective (Davids,92). The proposal is to extend the scope of the analysis to include the identification of expert(s) in the CSFs, highlighting the knowledge element(s) of the CSFs. To guide the identification process, a life-cycle description of the organisation's product i.e. the value-chain, is produced.

By identifying existing expert(s) and assessing their role in the CSFs, the knowledge element(s) in the organisation are highlighted at a strategic level. This would facilitate better understanding, on the part of the management, of the value of these experts to the organisation. It would also provide the basis for identifying potential KBS applications in the organisation that are relevant and meaningful (Beerel,87).

In the same light, the research proposes the development of a Knowledge Mapping Model (KMM). This model employs the matricing technique, which is in line with the models currently produced in IEM. KMM consist of three matrices. The first matrix cross-relates the various department's involvement against each objective, as identified in the organisation's business strategy, highlights the importance or relevance of existing knowledge elements in the overall business strategy. This would be enhanced with the results from the analysis of Goals and Problems, which has identified the various knowledge elements of each department. In the same manner, the second matrix cross-relates the "product life-cycle" against the various functions of each department, revealing the input of each department to the core business activity. This highlights the importance or relevance of certain activities to the core business activity. This would be enhanced with the results from the analysis of CSFs, by 'qualifying' the importance of certain activities with the "product life-cycle".

The two matrices are then merged to form a 'three-dimensional' matrix or view of the various knowledge elements in the organisation, against its business objectives and core business activity. The 'three-dimensional' model or KMM would provide the basis for further analysis and eventually, facilitate the possible deployment of relevant KBS in the organisation.

The resultant effect would be a strategic systems plan, consist of the KMM and the information systems plan. This will describe the necessary systems required for the organisation to achieve its overall business objective. In addition, existing knowledge elements are highlighted, providing the basis for identifying and developing KBS that are relevant and will offer real business advantages.

7. CONCLUSION.

Whilst KBS technology offers an opportunity for managing valuable knowledge and expertise, there has only been limited success in its use and deployment. One of the main reason for this slow uptake is the inadequacy of some of its systems development methodologies. The more established of these, KADS is limited in at least four areas: strategic planning, user-involvement, integration and future changes. In addition, the research has revealed that as a development methodology, IEM is quite adequate. However, its current scope is limited to conventional systems.

In light of this, the research work reported here propose to extend IEM in two ways. The first is to extend two techniques at the ISP stage, namely the Analysis of Goals and Problems, and the Analysis of CSFs, to highlight and spot knowledge pockets. The second is to develop a KMM, to enhance the information currently contained in one of its end-products, the information strategy plan.

These extensions are currently being implemented within a strategic systems project in a medium-sized engineering/manufacturing company located in the north-west region of the UK. Following a management buy-out, the company is currently undergoing a period of consolidation to review its operations, in particular the aspect of information provision. It currently operate on various out-dated software packages operating on antiquated hardware and is keen to 'seek' advantages through the use of 'up-to-date' computing technology.

8. REFERENCES.

Aktas(87). "Structured Analysis and Design of Information Systems", Prentice-Hall.

Ashworth(88). Structured systems analysis and design method (SSADM), *Information and Software Technology*, Vol 30(3), pp 153-163.

Avison & Fitzgerald(88). "Information Systems Development: methodologies, techniques and tools", Blackwell Scientific.

Avison & Wood-Harper(91). Information systems development research: an exploration of ideas in practice, *The Computer Journal*, Vol 32(2), pp 98-112.

BAe(90). "Development Methodologies for KBS: a Comparative Review", British Aerospace, UK.

Basden(89). A client-centred methodology for building expert systems, *in* "Computer and People V", Cambridge Press.

Battaglia(91). Strategic information planning: a corporate necessity, *Journal of Systems Planning*, February, pp 23-26.

Beerel(87). Identifying meaningful applications, *in* "Expert Systems: strategic implications and applications", Ellis-Horwood.

Boehm(88). A spiral model of software development and enhancements, *Computer*, May.

Bright et al(91). The KADS-II framework for KBS project management, *in* the proceedings of BCS-SGES workshop on KBS methodologies, 3rd & 4th December, London, UK.

Davids(92). "Practical Information Engineering: the management challenge", Pitman.

DTI(91). "KBS: Survey of UK Applications", Department of Trade & Industry, UK

Fitzgerald et al(85). Feature analysis of contemporary information systems methodologies, *The Computer Journal*, Vol 28(3), pp 223-230.

Hares(92). "Information Engineering for the Advanced Practitioner", John Wiley & Sons.

Hesketh & Barrett(90). "M1: An Introduction to the KADS Methodology", STC plc, UK.

Hickman et al(89). "Analysis for Knowledge-based Systems", Ellis-Horwood,

Hilal & Soltan(91). A suggested descriptive framework for the comparison of KBS methodologies, *Expert Systems*, Vol 8(2), pp 107-114.

Holloway(89). "Methodology Handbook for Managers".

Ince(88). "Software Development: Fashioning the Baroque", Oxford Science.

Journal(92). Special issue, *Journal of Systems Software*, Vol 19.

JTEC(94). Knowledge-based systems in Japan, *Comm. of the ACM*, Vol 37(1), pp 17-19.

KADS(92). "KADS bulletin", Touche-Ross, UK.

Khalr(91). Strategic implications of KBS methodologies, *in* the proceedings of BCS-SGES workshop on KBS methodologies, 3rd & 4th December, London, UK.

Lawson(90). Philosophies for engineering computer-based systems, *Computer*, December, pp 52-62.

Levine(85). Selecting a systems development methodology, *Infosystems*, 4 pp 72

Liou(92). Knowledge acquisition: issues, techniques and methodology, *Database*, Winter, pp 59-63.

Macdonald(86). Information engineering: an improved. automatable methodology for the design of data-sharing systems *in* "Information Systems Design Methodologies: Improving the Practice", North Holland.

Maddison(83). "Information Systems Methodologies", Wiley-Heyden.

Martin(82). "Strategic Data-Planning Methodologies", Prentice-Hall.

Martin(86). "Information Engineering Methodology" (4 volume-set), Savant, UK.

Melody(90). "Skills: cornerstone of the information economy", *CIRCIT newsletter*, 2(5).

Nonaka(91). The knowledge-creating company, *Harvard Business Review*, November-December, pp 96-104.

Richmond(92). Information engineering, the James Martin way, *Information Strategy: the Executive Journal*, Winter

Rock(93). Redesigning the future world, *Director*, August.

Scarff(91). GEMINI: a UK government initiative, *in* proceedings of the BCS-SGES workshop on KBS methodologies, 3rd & 4th December, London, UK.

Shie(93). Information strategic planning for knowledge-based systems, *in* the proceedings of the B.I.T. 93 conference, 10th November, Manchester, UK.

Shie & Sumiga(92) An evaluation of the KADS methodology, BCS-HCI workshop on "Expert systems in use", 1st April, Salford, UK.

Sprague & McNurlin(86). "Information Systems Management in practice", Prentice-Hall.

Thomas(91). What constitutes a KBS methodology, *in* the proceedings of the BCS-SGES workshop on KBS methodologies, 3rd & 4th December, London, UK.

Thomas & Adhami(91). Measuring the business benefits of expert systems, *Manufacturing Intelligence*, DTI No 9, Winter.

Waema & Walshman(90). Information systems strategy formulation, *Information & Management*, 18, pp 29-39.

Wasserman(83). Characteristics of software development methodologies, *in* "Information Systems Design: a Feature Analysis", North-Holland.

Whybrow(92). What to do and how to do it, *Informatics*, June.

Computer Supported Cooperative Work (CSCW)

JadeBird/III: A Collaborative Multimedia CASE Environment

Fu-Qing Yang[1], Wei-Zhong Shao[1], Wei Li[2]

[1]Dept. of Computer Science, Peking University, Beijing, 100871, P. R. China
[2]School of Computer Science, Florida International University, Miami, Florida 33199, USA

Abstract

JadeBird/III is the next generation of CASE environment, which combines the understanding of the way people work in groups with the enabling technology of computer networking, multimedia and associated hardware. The collaboration infrastructure in JadeBird/III provides collaboration among geographically dispersed multiple users of existing tools with minimal intrusion into existing software tools or user interaction style, and supports interoperability between a variety of collaborative applications. It can facilitate the development of software products with powerful support of an underlying multimedia repository which is a both syntactical and semantic uniform extension to standard SQL database server with a rich type of multimedia data, including real-time video and synchronized audio data. One of the salient features that makes JadeBird/III different from other computer supported cooperative environments is that the multimedia history information which consists of the specification, design, and rationale can be efficiently stored and retrieved /queried to support software redesign, reuse and evaluation.

1 Introduction

The information system development and production trends of 1990s are to reduce overheads, flatter hierarchies, shorten response time, and attend to customer requirements. These trends will continue to make dramatic demands on the information industry. In the face of such trends organizations are increasingly looking to: 1) improve communication between people; 2) reduce the time to make decisions, meanwhile improve the quality of decisions; 3) make rapid changes in development structure; 4) develop and modify products faster; and 5) reduce teamwork overheads and improve teamwork performance.

Over the years many basic tools and CASE environments have been produced to support the production of information systems. These include for example, IPSE2.5 [7], Software through Pictures [14], SoftBench [3], JadeBird/II [15] among many. However systems such as these are only part of the answer; simply having suits of tools which don't support collaboration well is not sufficient, particularly in the context of development of the larger and more complex systems. In the meantime, the support of group work has evolved naturally from a drive to increase personal productivity, and could lead to significant improvements in efficiency and cost-effectiveness of information system development [6]. Groupware is coined by Robert Johansen [5] as: *a generic term for specialized computer aids that are designed for the use of collaborative work groups*. Typically, these groups are small project-oriented teams that have important tasks and tight deadlines. Groupware emphasizes using the computer to facilitate human interaction for problem solving. Ellis *et al.* give an overview of the state of the art in [5]. Most existing systems are either almost completely concerned with providing support for the technical functions within the development process or provide relatively separate support for collaborative activities. This has the effect that such systems are not able to provide an appropriately distinguished collaborative working environment as there is little or no information available about the state of the collaborative process or powerful and integrated support for collaborative activities. Thus the idea of a computer based system aimed at supporting the versatile collaboration model rather than the use of computers as individual tools within the process is seen as an important concept. The application of these ideas to the complex area of information systems development is more unusual. The gains to be made in effectiveness are very important.

On the other hand, multimedia data have become increasingly popular and important in recent years due to advances in various hardware and software technologies such as faster CPU, wider communication bandwidth, bigger storage, audio/video capture/digitizing and playback devices with possible real-time compression and decompression, and standardization. Those advances are making it cost effective to integrate multimedia into workstations and desktops as in DVI [2], JMC-550 [12], *etc* . Research into computer and video fusion to support collaborative work has resulted in TeamWorkStation [11] which uses video-overlaid shared drawing surfaces in addition to wired video and audio communication links. Research into collaboration via multimedia facilities has resulted in systems like CECED [4], SHASTRA [1]. CECED (Collaborative Environment for Concurrent Engineering Design) provides mechanisms that facilities communicating effectively using multiple media and capturing the history of the informal phase of the specification and design process. It is designed to support collaboration among multiple users of existing tools. Strictly speaking, CECED is not a true *multimedia* computer supported cooperative environment because it incorporates only voice. SHATRA is another important collaborative design and scientific manipulation environment. Its intention is to provide a distributed heterogeneous environment for multimedia collaboration and to facilitate the development of collaborative applications. But its infrastrature impose many restrictions on tools integrated in the environment and it is very task specific.

These systems either provide simple collaboration (via audio/video communication), or content independent sharing of drawing and viewing surfaces, or are very task specific. In general, these systems have the following salient disadvantages:

1. The multimedia service in these systems is *ad hoc*. They do not have a integrated model for incorporating text, picture, sound and video. The multimedia communication messages can not be recorded and retrieved as part of history information of the system specification/design and decision-making. To support system redesign and

639

evaluation, it is necessary to capture, access, and reuse the information which consists of the design itself, together with rationale and development history. A successful capturing of history information will include these elements and answers to several questions: What is the design? How was the design achieved? Why were choices made? To this end, the database technology should be employed to capture, store, and query the multimedia entities related to the design and development history.

2. The collaboration infrastructure used in these systems is also *ad hoc* and imposes many restrictions on the tools integrated in the systems. In fact, the collaboration infrastructure should provide collaboration among multiple users of existing tools with minimal intrusion into existing software tools or user interaction style, support interoperability between a variety of collaborative applications, and support the co-existence of remote and local cooperation, personal and group working. The infrastructure must, in addition, facilitate the exchange of multimedia information which is useful to successfully communicate at the time of specification, design and development, and to share the results of tasks, and often necessary to actually solve problems.

Our intention in JadeBird/III is to provide a distributed heterogeneous collaborative multimedia CASE environment, and facilitate software development with powerful support of an underlying multimedia repository, which is both a syntactical and semantic uniform extension to standard SQL database server with multimedia elements. JadeBird/III can be used in:

Collaboration information system environment it can help a working group agree what has to be done, allocate tasks and roles, and undertake specific group activities such as collaborative specifications and group designs. For example, the ability to obtain comments on the initial design from the end users in an organized way could be of immediate benefit in many information system developments.

Meeting room system it can support face-to-face meeting activities such as the presentation and manipulation of information, decision making and action review. Such system could enable a group to reduce the number of unproductive meetings; and to ensure that critical process such as strategic planning produce better results.

Desktop video conferencing it will enable two or more geographically separated people or groups to interwork using a common screen display, video images of the other people in separate windows on the screen, and an integrated voice connection.

The rest of the paper is organized as follows: In Section 2, the system architecture of the JadeBird/III, including the Collaboration Manager, Session Manager and Repository Manager is described. Section 3 is the conclusions.

2 The System Architecture of the JadeBird/III

Jade Bird /III is a highly extensible, distributed and collaborative computer-aided Software Engineering(CASE) environment, where a geographically dispersed collaborative software design team can co-specify, co-analyze, co-design, co-code and co-debug information system projects over a distributed heterogeneous network of workstations and desktops. It differentiates those functional areas of its ancestor and other computer supported cooperative systems, namely

Shared Workspace Support which entails the mechanisms for the underlying database system and representation of information and the applications or processes that create and manipulate this information. Workspace are much more than just information storage and management systems; they also include audio, video, the computer tools and mechanisms used in the workspace. The collaboration services provided by *Shared Workspace* include:

- **Shared Tool Control** Control of a common set of tools that are running on each user's workstations is shared. Because the majority of tools used by designers provide single-user control, a special algorithm is needed to pass the control between users, ensuring that input from only one user at a time is received by the control.
- **Shared Remote Pointer** During a collaboration, remote users may want to indicate specific displayed items to supplement an audio dialog. A remote pointer may be realized by providing a separate pointing icon or by enabling control of the normal mouse/cursor.
- **Data Access** In addition to control of the tools, users may want to protect the persistent data that they own from modification by other collaboration participants. This additional protection service would be needed only when shared tool control is allowed.

Integrated multimedia Support which includes the modeling, presentation, representation, and manipulation of multimedia data. Real-time presentation and manipulation of video, audio, and image/graphics are also fully supported.

JadeBird/III supports collaboration among multiple users of existing tools with minimal intrusion into existing software tools. So, we can inherit the numerous software tools developed for the JadeBird/II without much change. The system architecture of the JadeBird/III is depicted in *Figure 1*.

At its core are a powerful Collaboration Manager which support synchronous multi-users tools, and a multimedia database which maintains and manages data entities or objects and relationships among them produced during the development of information systems. The Collaboration Manager is responsible for implementing the algorithms required for workspace sharing among users and to dynamically mediate critical resources among users and applications to avoid ambiguous or unintended results. The main purpose of a Collaboration Manager that supports concurrent access to applications is to establish and terminate multiuser dialogs, and to control information flow and synchronization among the participants of such dialogs. The Collaboration Manager accepts local and remote inputs, uses a resource contention resolution algorithm ,also called as *floor control algorithm* , to filter inputs, and then passes it to the applications.

The JadeBird/III also includes a comprehensive set of interoperable tools for software specification, design and code/debug. It provides an unified framework for collaboration, session management, data sharing and multimedia communication along with a powerful multimedia database. The environment provides a mechanism to create remote instance

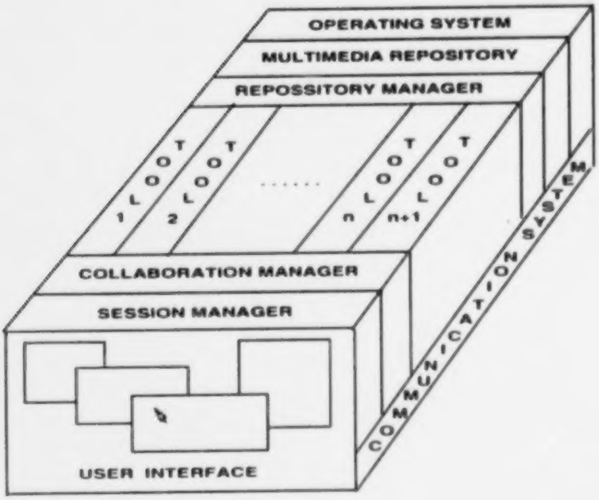


Figure 1: The System Architecture of the JadeBird/III

of applications and connect to them in client-server mode. It can be homogeneous collaboration (multiple instances of the same application) and heterogeneous (cooperating instances of different applications). In addition, the environment provides support for a variety of multi-user interactions from master-slaver electronic blackboarding to simultaneous multiple-users interaction. The Session Manager provides a mechanism for starting and terminating collaborative sessions, and for joining or leaving them. The tool processes connect to each other and communicate via the environment. The JadeBird/III provides a multimedia communication facilities enabling tools to use facilities and functionalities provided by other tools integrated in the environment.

2.1 The Communication System

During collaboration, users and multimedia applications can communicate with audio and / or video. The Communication System (CS) allows the establishment of audiovisual communication links

- between all collaborators for distributing the input obtained from a microphone or a camcorder at a user's workstation to all other collaborators;
- between a multimedia application and all collaborators for sharing the application's audiovisual input/output.

For communication between participants, the CS can establish a multiple to multiple relation in which each participant sends the source data (from the microphone and camcorder) to all other speakers and screens. The collaboration group is fully connected in its audiovisual input/ output. For application-specific data, the CS establishes a one to multiple relation in which the application is connected to all participants, application output is sent from the application to users. In both relations, a token controls the right to send data. In a multiple to multiple mode, only the token holder has the right to send data to the other collaborators. In the one to multiple mode, the token refers to the channel where input is provided to the application. Only the token holder can provide this input. Since the participants of a collaboration may have different audiovisual capabilities, the CS can obtain this information in order to establish the best possible quality link between each individual pair of communication partners. Once the video/audio link is set up, the sender digitizes audio/video signals from the microphone/camcorder and sends it over the established network connection. The receiver decodes the data into analog signals and send them to speakers /screen. These services can be built on local multimedia systems like IBM's Multimedia Presentation Manager [10], Microsoft's Video for Windows [13], or JMC-550 Real-time Multimedia System for Windows [12].

2.2 The Collaboration Manager

The Collaboration Manager monitors, intercepts, and manipulates the X protocol dialog normally conducted between a tool and the X server to provide collaborative presentation sharing (tool output) and key/mouse events (tool input). The collaboration architecture in the JadeBird/III uses a replicated computation model for the multiple users system, i.e., an instance of the application runs at each site involved in the collaboration. This model gives a performance advantage over a centralized model. The Collaboration Manager supports synchronous multiuser access to unmodified singer-user tools developed for the JadeBird/II by managing tool input/output from/to keyboard, mouse, and display. It dynamically mediates critical resources among users and tools to avoid ambiguous or unintended results, for example, ensuring that input to single-user, unmodified tools is generated by one user at a time.

In choosing a design model for the Collaboration Manager, we have several approaches: for example, *sharing between Xlibs, window managers, pseudo-servers, and X servers*. For the reasons of implementation and performance, we have chosen the *distributed pseudo-server architecture* in which sharing occurs between replicated pseudo-servers. This approach does not intrude on the applications or the original X server and eliminates the single-point collaboration bottleneck. A sharing pseudo-server has access to all client-server communication and window hierarchy information but does not require applications reloading. This implies that a comprehensive set of software tools developed for the other systems can be reused in the environment of the JadeBird/III without much modification. The only work needed to do is, starting the pseudo-server as their X server for all applications which might be included in the collaboration. *Figure 2* shows this in more detail.

The Collaboration Manager permits the use of unmodified single-user applications developed for the JadeBird/II in a new multiuser environment. The potential exists for multiple users to generate input to an application simultaneously that the unmodified applications is not prepared to handle. By applying a floor control policy, the Collaboration Manager can ensure that input to a single-user unmodified application is generated by only one user at a time.

This only-one-input-at-time floor control policy determines whether an individual can generate input to an application, but does not distinguish among individuals or their roles. It is sometimes desirable to limit or prohibit a user from generating input based on the

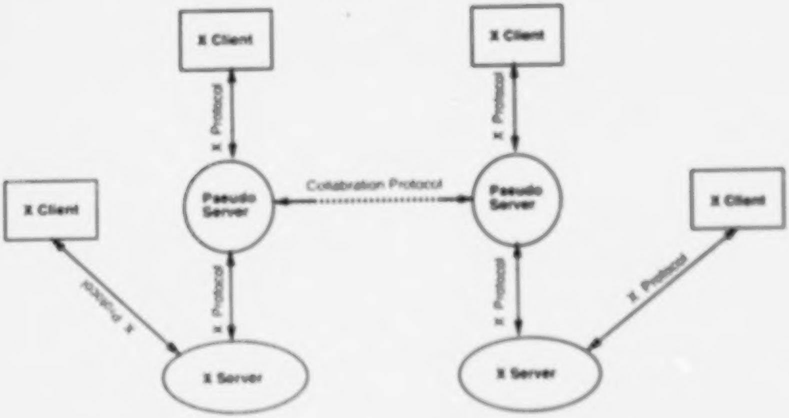


Figure 2: *The Model of the Collaboration Manager*

identity or role of an individual. For example, when peer designers are comparing results, each designer may view but not modify results shown by another designer. By applying the access permission control mechanism to a floor control policy, the Collaboration Manager can control who may affect the applications, in addition to controlling the number of participants who can simultaneously access the application.

A two-tiered permission based access mechanism is used to structure a variety of multiuser interaction modes at run time. Permissions are specified on a per-site as well as per-object basis. A site permission defines the scope of collaboration operations available at each site in the session. An object permission specifies the operations permitted on each object in the collaboration.

2.3 The Session Manager

A collaboration session in the JadeBird/III is started by a user called session leader through a Session Manager. One instance of a Session Manager runs per collaboration. The Session Manager is a specialized component in the environment. It maintains the collaboration and handles details of connection and session management . Similar to window managers, the Session Manager allows tools and the Collaboration Manager to be independent from the common, collaboration-related services it provides such as open and close of tool user interfaces. A collaboration tool must advertise its window identifier and name information to the Session Manager so that Manager can access its window and provide a visual representation by which the other collaborators may access this tool. The Session Manager has a constraint management subsystem which resolves conflicts that arise as a result of multiple user interactions over the shared objects, and therefore maintains mutual consistency of multiple operations. The Session Manager also coordinates and manages the establishment of multimedia multiparty connections among applications on different workstations. It facilities negotiation among participants regarding the configuration of the shared environment , and passes configuration and state information to the Session Managers on the other sites.

A typical collaborative session in JadeBird/III is depicted in *Figure 3*.

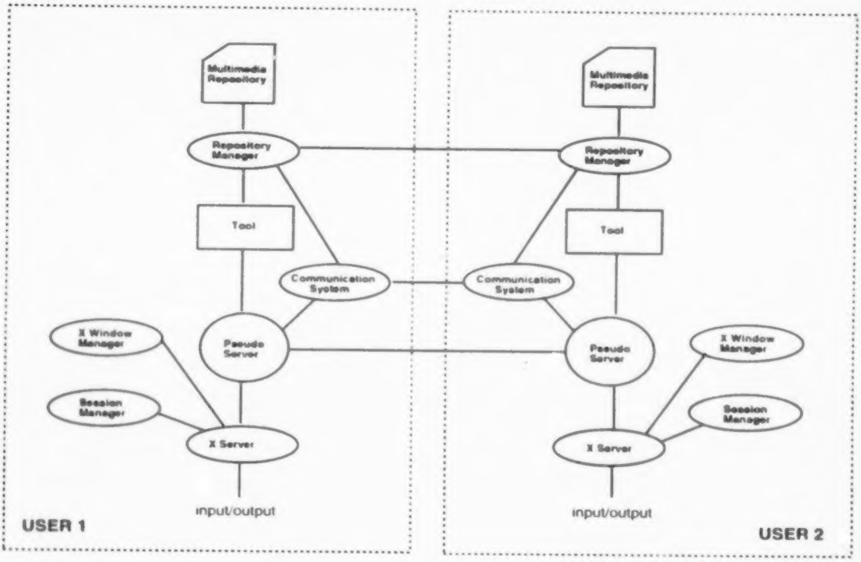

Multimedia Repository
Repository Manager
Tool
Communication System
X Window Manager
Pseudo Server
Session Manager
X Server
input/output
USER 1
Multimedia Repository
Repository Manager
Tool
Communication System
Pseudo Server
X Window Manager
Session Manager
X Server
input/output
USER 2

Figure 3: *A Collaboration Session in JadeBird/III*

2.4 The Repository Manager

The Repository Manager coordinates collaboration access to persistent information by providing an access control and locking mechanism. It mediates between tools and collaboration-unaware multimedia information storage system (Repository) to ensure data consistency, data location transparency, and tool state consistency. The Repository Manager acts as a multiplexor for all data access operations, after receiving multiple duplicate requests from its peers, it issues a single request to the Multimedia Repository and returns the response to each requestor. To ensure consistent data and tool state, the Repository Manager takes responsibility of the transition of user information or objects involved in collaboration from private workspace into the shared workspace, and back. The Repository Manager does not really copy the private data into the shared workspace, because this it will result sever data redundancy, and lead to potential data inconsistency. In the JadeBird/III, the Repository Manager modifies the access control list on private objects to accommodate group access. It does this by defining a special user identity for a collaboration and adding it to existing access control list for the private objects, i.e., use the name of the collaboration as the user name to access the objects. It then maintains a mapping between active members of the collaboration and the name of the collaboration. When objects in shared workspace are removed back to private space, or when the collaboration is terminated, the Repository Manager deletes the collaboration name from the private object's access control list.

2.5 The Multimedia Repository

The Multimedia Repository (MR) is an extension to SQL database server which provides database support to the integrated environment as well as to the modeling, presentation, representation, and manipulation of multimedia data. The focus of the extension is placed on video and audio data, because of the specific requirement of multimedia collaboration activities. The interested readers can refer to [8, 9] for more details about multimedia extension to the SQL database server.

3 Conclusions

At present, the work is being done at both Peking University in China and Florida International University in USA.

The Collaboration Manager and Session Manager are being designed and prototyped on SUN workstations. the infrastructure of those two managers will determine the overall structure of the environment. A systematic and effective design will be worked out by borrowing many techniques we have developed for the series of JadeBird products.

In the meanwhile, study on the collaboration process model will be undertaken, which is central to the problem of improving collaboration group efficiency from the theoretical point of view. This study will focus on cooperative activities that will benefit from technological advances in multimedia workstations and high-speed networks. Activities to be considered include shared design and document -handling, monitoring of process control and team tasks. The study is to investigate the effects on multimedia architectures of different temporal, spatial and organizational constrains. The study will start by reviewing existing research and establishing methodologies for investigating remote cooperations.

References

[1] Vinod Anupam, *et al.* , " Collaborative Multimedia Scientific Design in SHASTRA", *Proc. First ACM Int'l Conf. on Multimedia*, Anaheim, CA, August 1993, pp.447-480.

[2] Bunzel, M. and Morris, S. *Multimedia Applications Development Using DVI Technology*, Addison-Wesley, Reading Mass, 1992.

[3] Cagan, M. R. " The HP Softbench Environment : An Architecture for a New Generation of Software Tools", *Hewlett-Packard Journal* , 41(3):36-47, June 1990.

[4] Earl Craighill, *et al.*, " CECED: A System For Informal Multimedia Collaboration", *Proc. First ACM Int'l Conf. on Multimedia*, Anaheim, CA, August 1993, pp.437-445.

[5] Ellis,C., *et al.* , " Groupware: Some Issues and Experiences", *Comm. of the ACM*, Vol.34, No.1, Jan., 1991, PP38-58

[6] Prasun Dewan, and Johan Riedl, " Towards Computer-Supported Concurrent Software Engineering" Technical Report, Purdue University, 1993

[7] Snowdon,R. A. , " An Introduction to the IPSE2.5 Project " International Workshop on Environment, Chinon, France, September, 1989

[8] Sha Guo, Wei Sun, Wei Li, *et al.*, " Panther: An inexpensive and Integrated Multimedia Environment", IEEE International Conf. on Multimedia Computing and System. Boston, May, 1994.

[9] Sha Guo, Wei Sun, Wei Li, *et al.* , "MSQL: An SOL-based Relational Database Extension to Support Multimedia Data", the 5th International Hong Kong Computer Society Database Workshop. Feb., 1994

[10] IBM, " Multimedia Presentation Manager/2: Programming Reference" IBM Document, 1992

[11] Ishii, H., Miyake, N., " Toward an Open Shared Workspace: Computer and Video fusion approach of TeamWorkStation", *Comm. of the ACM*, Vol. 34 No. 12, Dec. 1991, PP36-49

[12] Milky Way Computer Corp., *JMC-550 Image/Video/Audio Real-Time Compression Board User Manual*, 1993.

[13] Microsoft Corporation, "Video for Windows Development Kit Programmer's Reference", Microsoft, 1993

[14] Wasserman, A. I. " Tool Integration in Software Engineering Environment", International Workshop on Environment, Chinon, France, September 1989

[15] Yang Fuqing, Shao WeiZhong and Li Wei " A CASE Environment of Jade-Bird/II" Chinese Journal of Electronics, Vol.2, No.3, Oct, 1993

Semantic Model of Asynchronous Distributed Cooperation

I.T. Hawryszkiewycz[1], **L.A. Maciaszek**[2], **J.R. Getta**[3]

[1]University of Technology, Sydney, Australia
[2]Macquarie University, Sydney, Australia
[3]Wollongong University, Wollongong, Australia

The trend towards an information-based society calls for computers to assist people to collaborate across time and space. Such collaboration is now frequently needed to bring expertise together to develop artifacts such as documents, project plans or other designs. Computer support for asynchronous distributed cooperation must allow individuals to work on related artifact components for short periods on their own and then bring their work together through transformations that are consistent across the related components and agreed to by the collaborators.

This paper defines a conceptual framework for asynchronous distributed cooperation. The proposed semantic model is essentially an abstraction of our earlier implementation model, named RESPONSE, and it derives its expressiveness from object-oriented modeling. It is also an extension of our earlier work on artifact semantics. The paper relates the semantic model to the RESPONSE implementation model and to the concepts drawn from the activity theory. The semantic model describes an integrated conceptual framework for artifact and coordination semantics.

1. INTRODUCTION

The acronym **CSCW** (Computer Supported Cooperative Work) is used to describe a group of people working together to achieve a common goal using a customized collection of computer technologies (Schmidt and Bannon, 1992). The common goal is usually an artifact development (e.g. a document or design). An artifact is a computer-stored object that contains data and associated knowledge, such as policy rules, protocols, assumptions, comments, observations, operations. The CSCW technological support is usually known as *workgroup computing*. It combines a range of diverse technologies, such as electronic mail, document and multimedia management, spreadsheets and database systems, etc.

The CSCW applications have introduced a new requirement into ***artifact semantics***. This requirement is to move from the idea of sharing data to jointly owning it. *Joint ownership* implies that team members collaborate on parts of an artifact and negotiate decisions on changes to its content and structure. An artifact can assume different *states* (such as working, pending, and released) and can have many *versions* in each of the state. Changes between states and versions are affected through structured negotiability between collaborators. We use the term *configuration* to define meaningful compositions of artifact versions in a given state. Configurations constitute units of collaborative work as selected by individual collaborators.

An underlying feature of cooperation is that it happens by following some coordination rules. Hence, apart from artifact semantics, CSCW applications must support ***coordination semantics*** through coordination rules. Some of these rules are predefined and deterministic, others are unanticipated and non-deterministic. Coordination is a necessary condition of collaboration, but otherwise these two concepts are independent. For example, traffic lights impose some coordination on the traffic without forcing the drivers to collaborate through some communication channels. On the contrary, the joint authorship of a document requires a *coordinated collaboration* among the authors. In general, *coordination* is a set of rules (or protocol) that defines the *collaborative process* needed to develop a jointly owned artifact. The collaboration is achieved through *interaction* between the participants using allowed *operations* on artifact objects.

In this paper, we concentrate on a major class of CSCW applications which are characterized by ***asynchronous distributed interaction***, i.e. different-time/different-place (or any-time/any-place) applications (Ellis *et al.*, 1991). This class of applications places a strong demand on the correctness and completeness of the definition of coordination semantics and on the integration of coordination and artifact semantics. An asynchronous distributed CSCW system must integrate coordination semantics into workspaces of people with minimal impact on individual work. Coordination is realized by connecting workspaces, each of which can support one or more users. Artifact objects can be moved between workspaces through workspace connections using specified coordination mechanisms. Coordination semantics supports joint decisions on artifact changes and ensures that all collaborators are aware of artifact development status.

The paper extends our earlier work (Hawryszkiewycz, 1993) on a generalized CSCW system by including artifact management semantics. It also extends our latest work on CSCW implementation model (Hawryszkiewycz *et al.*, 1994) by providing a semantic model for collaboration in asynchronous distributed systems. The core of our CSCW implementation model have been successfully verified through the implementation of the prototype of a CASE tool for Object Database Application Modelling (Tjhin, 1993). This prototype has given us a direct evidence that the semantic model proposed here is implementable under an Object Database Management System (VERSANT in our case).

2. SEMANTIC AND IMPLEMENTATION MODELS OF CSCW

In Hawryszkiewycz *et al.* (1994) we argued that CSCW is sufficiently diversified to preclude the possibility of developing a single implementation model, no matter how abstracted, for various CSCW applications. We proposed that a CSCW implementation model should be based on an object database system. We then concluded that while implementation models need to be requirements (i.e. application) specific, they can be generalized for particular classes of systems, rather than to individual applications. We finally proposed an implementation model for a CSCW class of systems characterized by asynchronous distributed interaction. The model was given a broad acronym **RESPONSE** (**RE**quirements **SP**ecific **O**bject **N**etwork **S**ystem **E**nvironment). This paper specifies the semantic model of RESPONSE for asynchronous distributed interaction. However, being a high-level abstraction, our semantic model can be treated as independent from the RESPONSE implementation model.

Figure 1 illustrates the RESPONSE concept and the components of RESPONSE models. Asynchronous distributed interaction is one possible **re**quirements **sp**ecific **o**bject **n**etwork **s**ystem **e**nvironment (Ellis *et al.*, 1991). The ***implementation model*** (Hawryszkiewycz *et al.*, 1994) describes three architectural components:

- the process management, which defines the processing required for coordination or workgroup computing activities;
- the artifact management, which defines the artifact structure and access;
- the presentation interface, that defines this portion of the computer interface which is characteristics to all applications within the class of asynchronous distributed applications.

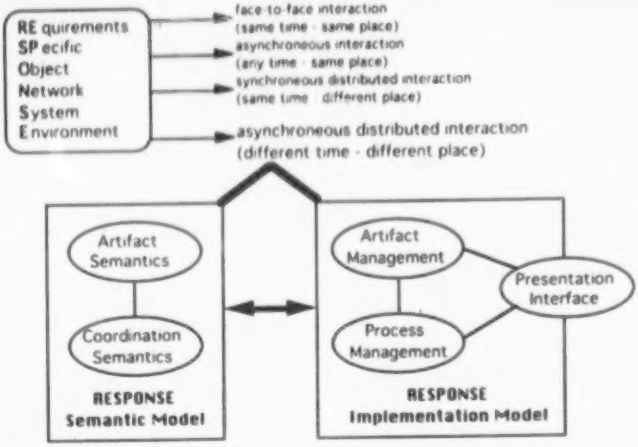

RE quirements
SP ecific
Object
Network
System
E nvironment
face-to-face interaction
(same time - same place)
asynchroneous interaction
(any time - same place)
synchroneous distributed interaction
(same time - different place)
asynchroneous distributed interaction
(different time - different place)
Artifact Semantics
Coordination Semantics
RESPONSE Semantic Model
Artifact Management
Process Management
Presentation Interface
RESPONSE Implementation Model

Figure 1. RESPONSE models

The ***semantic model*** consists of two tightly integrated components:

- the artifact semantics, that describes the conceptual structure of artifact objects and interdependencies between object components, states, versions and configurations;
- the coordination semantics, that defines the primitives for actions on artifact objects and for coordination rules between collaborators.

The challenge is to maintain a relationship between the components. The artifact management must support the semantics of joint ownership and allow collaborators to work independently and without interference on artifact objects but reach agreement on any permanent changes to the artifact. The process management must support the coordination rules inherent in such collaboration. The interface must be able to present the artifact to the collaborators but also make them aware of the actions of other collaborators. The RESPONSE interface for asynchronous distributed interaction is nicknamed WYDIWIS (What You Did Is What I See).

3. SEMANTIC CONCEPTS OF THE RESPONSE MODEL

The generic concepts in our semantic model (Hawryszkiewycz, 1993b) have drawn from both theoretical and empirical work. The model uses ideas from activity theory proposed by Kuutti and Arvonen (1992) as well as those drawn from ontology (Wand and Weber, 1993). As shown in Figure 2, activity theory views cooperative activities as a group of active subjects working on a joint object supported by information technology. The important words here are ***active subjects*** and ***joint object***. The word active implies that the subjects themselves can determine the course of action as well as changing the structure. In contrast passive subjects merely react to a request in a prespecified manner. The word joint rather than shared is used to imply common ownership. Thus the whole group is interested in the state of the whole object and may jointly decide on the future evolution of the object rather than simply using parts of that object for their own needs. A whole organization may be made up of a number of such groups and these groups may also be connected. Thus coordination exists both within groups and between groups.

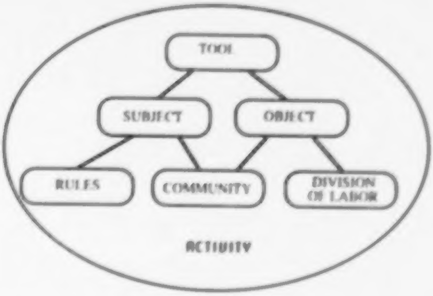


Figure 2. Activity Theory as applied to CSCW

The concepts of tool, subject and object are sufficient to describe the simple concept of work. The model in Figure 2 extends these basic concepts to place them into the context of a wider organization. The first extension is to define a community within which the work proceeds. Thus all of the subjects will come from the community, and the community as a whole can show interest in an object. The idea of rules is to specify the behaviour of subjects in the community. Thus the community can set the rules by which subjects must conform. These rules will define the interactions between subjects in any collaborative work. The division of labor is again defined by the community and assigns work on objects to community members.

Using the structure described in Figure 2 as a guideline, we propose the following set of CSCW semantic concepts:

Artifacts - Artifacts represent the organization's information base. These can include files as well as artifacts such as reports, designs and so on. The artifact can include data values as well as methods to operate on these values, constraints and rules about the values, and policy rules used in the decision making process or in selecting appropriate roles to perform tasks.

Roles and Actors - Roles are abstract entities that represent system decisions. For example, purchasing a part requires a purchase requisition to be made and approved. The requisition and approval decisions are modeled as two separate roles. An actor is assigned to each role to make the decision. Actors are often positions that can be derived using organizational rules. Roles are not permanently associated with positions but are dynamically assigned using organizational rules. Responsibility changes can thus be made by only changing an organizational rule and not the system structure. A person can have thus many roles and there is nothing to prevent a person from transferring knowledge (informally) between roles as often happens in organizations.

Tasks - Task objects carry out some well-defined function in a problem domain and are generally of shorter duration than environments. They can be parts of a complex transaction or simple work tasks such as editing a document.

Environments - Environments model the support structures for groups. Environments can be of prolonged duration and support a variety of processes (known as activities) each composed of tasks and roles. These processes can combine the tasks objects and roles into group procedures. Environments can include other environments, as well as tasks, roles or data. An environment must include at least one role and usually covers one group and it is often centered around one artifact or a set of related artifacts.

Activities - Each activity is within an environment and primarily models interactions between objects in the environment. They determine the interaction of roles and tasks within a group. They are realized by coordination objects, which define the communication patterns between objects in the activity. Activities can only include objects from their environment.

Ports - Ports model connections between objects. They can be of a number of levels of complexity (Hawryszkiewycz, 1993a) ranging from basic ports to speech act ports (Searle, 1969) and to conversation ports, where speech acts are composed of basic ports and then conversations are synthesized from speech acts. Thus roles or groups can be empowered with ports, which give them certain coordination capabilities.

4. EXPLANATION OF CHOICE

Our choice of objects can be related to *activity theory* as well as *ontological theory*. In the ontological context, our choice is one where the data objects predominantly correspond to the physical ontological elements while behaviour is modeled using state transitions between objects.

The proposed concepts are also consistent with the argument put forward by Kuutti and Arvonen (1992) that defines cooperation as "A set of multiple active subjects sharing a common object, supported by information technology". In our case the active subjects are the roles, the objects are the artifacts and tasks provide the tools.

The concept, *environment*, is an extension of Kuuti's model as it gives us the ability to group and manage definitions of interactions. In Kuutti's model all the interactions between subjects are described as one whole. The idea of splitting Kuutti's activity into environment and activity in our model gives us an opportunity to 'modularize' the interactions. We thus see the environment as corresponding to Kuutti's activity, and then use our activities to group interactions into semantically meaningful groups. Environments are flexible and can be expanded to create other environments that include their own roles, tasks, artifacts and activities, thus further improving modularity and providing the ability to model larger systems.

Another specific choice made in the model is to distinguish between role and actor. A *role* identifies a logical focus of responsibility, whereas the *actor* is the physical agent that carries out this responsibility. A number of other modeling methods simply use the concept of agent as the term to indicate the existence of an active subject, but not making a distinction between the logical and physical object. In that case the agent is both a role and actor. We make the distinction for two reasons. One is the dynamic nature of agents in practice, where in real life different people act as agents in more than one application. The concept of actor allows all of one particular person's responsibilities together. Another reason is that in practice the physical actor can dynamically change. We thus make the distinction specific and can adaptively assign actors to roles. This is also similar to the idea of rules existing between subject and community in activity theory - in this case the subject is the role whereas the rule selects a particular actor for the role.

In our prototype implementation (Huang, 1992) each object is represented by a workspace, which becomes a window on the screen. The object window contains the objects in its context as well as the properties, rules and methods applicable to the object. The object contents can be selected using the window menus. We describe how such objects can be combined to realize semantics commonly found in work environments.

Our concepts can be related to other prototypes. The concept of environment also appears in ConversationBuilder (Kaplan *et al.*, 1992) and is to some extent similar to an OVAL's

(Malone *et al.*, 1992) view. We also make a clearer distinction between environment and activity to allow us to model a variety of cooperating processes within the same environment. Many other systems, as for example ConversationBuilder, also use the term activity in the same way as we do. However, we make a further clear distinction between activity and coordination. Activity describes the collection of environment objects that work together to achieve some goal whereas coordination defines the processes used in the activity. The separation of coordination and activity also increases independence by allowing generic coordination patterns to be defined independently of activities.

Coordination needs some further explanation as it is central to modeling system dynamics. Dynamics are modeled through object states. The general semantic here is that a state value triggers an object method. The completion of a method becomes an event which serves as a trigger to another object. An activity can include any number of objects and events. Rules that specify such coordinations are stored in an activity. Alternatively a generic set of rules can be stored in *coordination objects*, which can be included in activities. Our model thus differs from some other models in the way that dynamics are modeled because the emphasis is on states rather than messages. This enables the definition of generic coordination objects, which define coordination by state transitions. Another goal is to identify generic coordination objects that can be reused in applications.

Generic coordination objects do not appear in most systems. OVAL, for example, uses an approach where agents operate on folders that receive messages. An agent can be triggered by the arrival of a message. Following agent action, the agent may move a message into another folder, thus triggering a further agent. Coordination is thus distributed between agents and no facilities for generic coordination are described. ConversationBuilder treats activity and coordination synonymously, referring to the coordination process as a protocol but not indicating whether generic protocols are supported.

In the table below, we relate the concepts of the RESPONSE semantic model to those used in the RESPONSE implementation model and to Kuutti's model.

RESPONSE conceptual model	**RESPONSE implementation model**	**Kuutti's model**
artifact	workspace, class and instance object, composite and/or versioned object	object
role and actor	workspace and user	subject, community
task	session, short transaction	work on an artifact
environment	CSCW application	activity
activity	long transaction, session	coordination rules, division of labour
port	channel	-

5. FRAMEWORK FOR ARTIFACT SEMANTICS

Figure 3 illustrates the idea behind artifact semantics. The structure of an artifact is equivalent to the notion of object. An artifact is therefore an identifiable entity that is the subject of cooperative work. ***Artifact objects*** are acted upon by roles, actors, tasks and activities. Artifact objects can be *transient* (in which case they do not last beyond the duration of a task or an activity) or they can be *persistent* (in which case they are permanently stored in the environment, unless explicitly deleted). Artifact objects have *composite semantics*, i.e. they can recursively contain smaller artifact objects. The depth of the *semantic decomposition* is application dependent, although the *implementation decomposition* is almost always deeper and ends with so called *literal objects* (strings, integers, etc.). For example, the semantic decomposition of a document artifact may be Document ⇒ Chapter ⇒ Section, while the implementation decomposition of this artifact can extend further to include ⇒ Paragraph ⇒ Sentence ⇒ Word. In any case, the principle that "everything is an object" is adhered to, and extends on the other semantic notions of Figure 3, namely versions, states, configurations, and customizations.

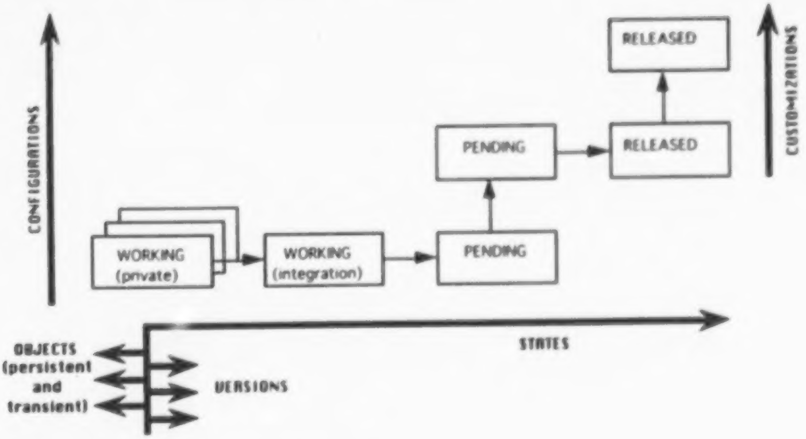


Figure 3. Artifact semantics in RESPONSE

Most , but not necessarily all, artifact objects are versioned, i.e. multiple ***versions*** of the same artifact object can be created, presumably by different cooperating roles. Once created, a version is given a unique identity and performs like any other artifact object, except that versions of the same object belong to the same *version history*. This means that the *version derivation tree* is maintained. Different or the same roles can work on multiple branches of versions, called *alternatives*. Alternatives can be later merged according to prespecified protocols (Hawryszkiewycz *et al.*, 1994).

Artifact objects, whether versioned or not, can change ***states***. One option supported in RESPONSE, are three states of cooperation: *working*, *pending*, and *released*. (However, the pre-determined identification of state hierarchy is more a matter of convenience, then a fixed concept.) States belong to specific *roles* and only the roles associated with a state can access artifacts in that state. The working state is further subdivided to distinguish the states owned by individual roles (*private* state) from joint states accessible to group roles (*integration* state). The coordination semantics, discussed in the next Section, governs the movement of artifact objects between states.

Artifact objects at the bottom level of semantic decomposition are the smallest possible units of work available to roles. However, the cooperation activity requires that bottom level objects are successively developed into larger artifact objects until the goal of the environment, i.e. the top level artifact, is achieved. This activity is supported through combining objects (versioned and unversioned) of the same state into ***configurations***. Since configurations are also objects they can be uniquely identified, invoked and acted upon.

While configurations are identified in all three states, they are restricted in the RESPONSE model to a single environment. However, environments themselves can be composed and can contain other environments. Thus the production of a joint report can be divided, say, among three environments responsible for the introduction, the proposal and the evaluation. Released configurations of these three environments constitute the final report. ***Customization*** is a configuration that integrates configurations released by a number of environments.

6. FRAMEWORK FOR COORDINATION SEMANTICS

In the RESPONSE model, an artifact evolves through three states: working, pending and released. Consequently, the coordination semantics defines primitive operations necessary to move artifact objects between these states. In general, the evolution of an artifact to the next state can be achieved through one of the two modes of coordination semantics:

(1) through negotiations between equal roles, or

(2) through overriding decisions by privileged roles.

Between these two ends of the coordination spectrum, multiple intermediate solutions are possible. As discussed in Section 4, generic coordination objects do not appear in most CSCW models reported to date. In other words, the first mode of coordination semantics is dominant. This is consistent with the emphasis placed in CSCW research on the existence of the direct communication line (port) between roles.

The RESPONSE approach to the coordination semantics integrates the two modes of coordination semantics. We allow negotiations between roles through the concept of *active objects* whereby an artifact object can trigger execution of coordination operations. Since one such operation can be to broadcast messages to roles, active objects can exhibit the coordination semantics of negotiations between *equal roles*, if so desired. We also support roles with *privileges* to change artifact states. Only actors with such roles can transform an artifact from one state to the another. This way, we achieve one of the principal objectives of the RESPONSE model - to seamlessly integrate the artifact and coordination semantics.

In this paper we identify four artifact states, and need three privileged roles to transform objects between these states:

- integration privilege,
- pending privilege,
- release privilege.

Furthermore a role with pending privilege has also integration privilege, and a role with release privilege has also the other two privileges. Only those actors who take these roles can initiate state changes of the artifact.

Figure 4 illustrates the coordination semantics of RESPONSE. The four roles are shown together with three workgroup states through which artifact objects move. (The workgroup states can be implemented as server object bases or can be distributed among workspaces of actors.) The lines between states represent RESPONSE ports. Ports carry operations which affect artifact states. There are only four primitive operations:

- sendmessage(),
- copy(),
- checkin(),
- checkout().

655

Actors who take role A1 in Figure 4 have no privileges to upgrade artifact states. Artifact objects created by these actors can only be copied to the private workspace of actors who take role A2 that has the integration privilege. This process is facilitated by a direct port between these two actors which supports negotiations between equal roles.

Actors who take role A2 have an integration privilege and can checkin an artifact object to the integration workspace. The integration workspace combines the in-progress work of a group of actors. The relationships between the old (parent) object version and the new (checked-in) version are automatically maintained by the system by traversing the version derivation graph.

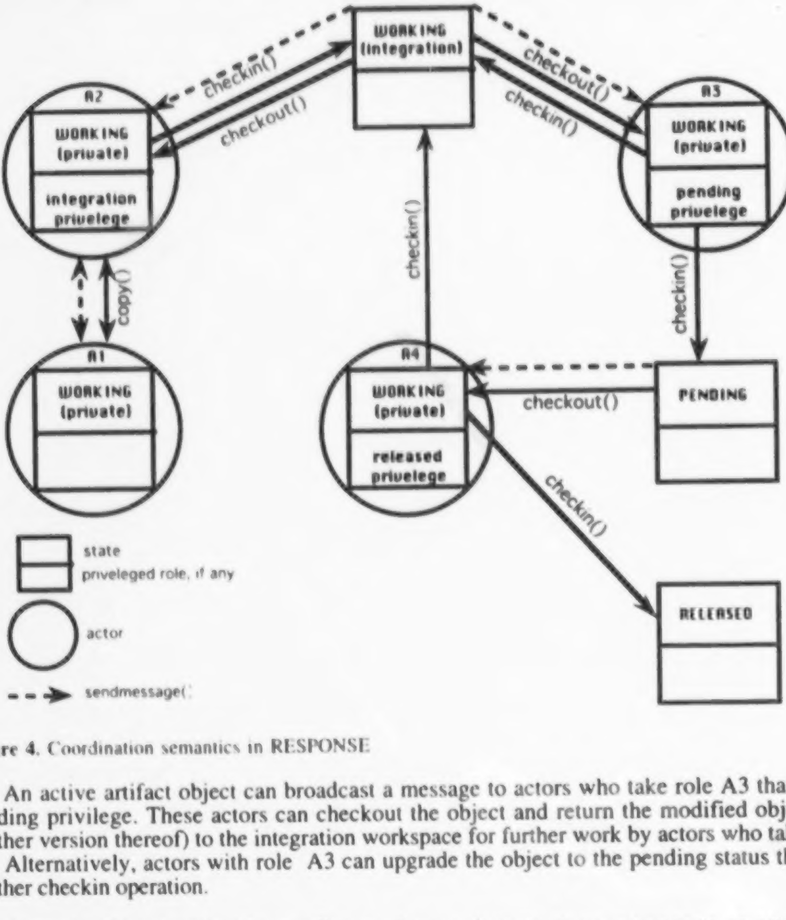

WORKING (integration)
A2
WORKING (private)
integration privelege
A3
WORKING (private)
pending privelege
A1
WORKING (private)
A4
WORKING (private)
released privelege
PENDING
RELEASED
checkin()
checkout()
checkout()
checkin()
checkin()
checkin()
copy()
checkout()
checkin()
state
priveleged role, if any
actor
sendmessage()

Figure 4. Coordination semantics in RESPONSE

An active artifact object can broadcast a message to actors who take role A3 that has a pending privilege. These actors can checkout the object and return the modified object (or another version thereof) to the integration workspace for further work by actors who take role A1. Alternatively, actors with role A3 can upgrade the object to the pending status through another checkin operation.

Once the object reaches the pending workspace, it generates a message to an actor who take role A4 that has the release privilege (A4) and can check out the artifact. An object checked out from the pending workspace can either be *promoted* by being checked in to the release workspace or *demoted* by being checked in to the integration database (i.e. the object cannot be directly updated). Once demoted, the object can be changed or a new version of it can be created and then it can be checked in again to the pending workspace, thus closing the cooperation loop. The RESPONSE system is responsible for deleting objects from the pending workspace once they are promoted to the released status.

The release workspace contains objects of a RESPONSE project that have been fully verified and approved. However, the actors with the release privilege can at any time demote a released object to an integration status, thus beginning another cycle of the collaboration loop.

7. CONCLUSION

In this paper, we have defined a semantic model for asynchronous distributed cooperation. We argued in Introduction that asynchronously-distributed CSCW environments require an integration of artifact and coordination semantics within the processing pattern which we call a coordinated collaboration. We then compared the semantic and implementation models of CSCW and attested that we achieved a satisfactory mapping between the two models in a prototype implementation of a CASE tool under an Object Database Management System. We proceeded with the definition of the concepts in our semantic model and explained our choice. Finally, we related and integrated the concepts into frameworks for artifact and coordination semantics.

REFERENCES

ELLIS, C.A., GIBBS, S.J. and REIN, G.L. (1991): Groupware: Some Issues and Experiences, *Comm. ACM*, 1, pp.39-58.

HAWRYSZKIEWYCZ, I. (1993a): Supporting Coordination in CSCW Systems, *Proc. Int. Symposium on Autonomous Decentralized Systems*, Kawasaki, Japan, IEEE Computer Society Press, pp.225-231.

HAWRYSZKIEWYCZ, I. (1993b): A CSCW Design Tool Based on Generic Concepts, *Proc. 12th Int. Workshop on Informatics and Psychology*, Schaerding, Austria, to be published by North-Holland.

HAWRYSZKIEWYCZ, I., KARAGIANNIS, D., MACIASZEK, L. and TEUFEL, B. (1994): RESPONSE - Requirements Specific Object Model for Workgroup Computing, *Int. J. of Intelligent & Cooperative Information Systems*, 21p. (to appear Sept. 1994)

HUANG, Y. (1992): *SPOOCS: An Object Oriented Open System Prototype*, Research Report 92.6, Key Center for Advanced Computing Sciences, University of Technology, Sydney, 36p.

KAPLAN, S.M., TOLONE, W.J., BOGIA, D.P. and BIGNOLI, C. (1992): Flexible, Active Support for Collaborative Work with ConversationBuilder, *Proc. CSCW'92*, Toronto, Canada, ACM, pp.378-385.

KUUTTI, K. and ARVONEN, T. (1992): Identifying Potential CSCW Applications by Means of the Activity Theory Concepts: A Case Example, *Proc. CSCW'92*, Toronto, Canada, ACM, pp.233-240.

MALONE, T.W., KUM-YEW, L. and FRY, C. (1992): Experiments with OVAL: A Radically Tailorable Tool for Cooperative Work, *Proc. CSCW'92*, Toronto, Canada, ACM, pp.289-297.

SCHMIDT, K. and BANNON, L. (1992): Taking CSCW Seriously, *Computer Supported Cooperative Work, An International Journal*, Vol.1., No.1, 1992.

SEARLE, J.R. (1969): *Speech Acts*, Cambridge University Press, Cambridge.

TJHIN, A. (1993): *A CASE Tool for Object Database Applications Modelling*, MSc Thesis, Macquarie University, Sydney.

WAND, Y. and WEBER, R. (1993): On the Ontological Expressiveness of Information Systems and Design Grammars, *J. Inf. Syst.*, Vol. 3, No. 4, pp.217-238.

COOPERATIVE WORKING FOR END-USERS WITH MMTCA

Sandy Weydd[1] and Malcolm K. Crowe[1]

[1] University of Paisley, Dept. of Computing and Information Systems, Paisley PA1 2BE, Scotland

Abstract

One of the main problems with CSCW systems has been to persuade users to take up the concepts of groupware products in sufficient numbers to provide the critical mass that a CSCW application needs to be effective. A main stumbling block for this has been the fact that many CSCW systems only use a restricted toolset of applications specifically designed for group work, meaning that the user cannot utilise the applications which they already have and are comfortable with.

The Multimedia Toolbox for Cooperative Applications (MMTCA) is an ESPRIT project which aims to produce an environment which will allow users to integrate their existing single-user applications into a cooperative framework, and to allow users to design their own cooperative applications to automate repetitive administrative work processes. This paper describes the concepts behind the MMTCA system, gives an overview of the architecture and design, indicates the progress of work so far, and how it relates to problems which have been encountered by users of CSCW systems.

1. INTRODUCTION

1.1. Problems in CSCW Systems

Computer Supported Cooperative Work (CSCW) is a field of research which has been around for several years, and systems which fall into this category have been designed and put into use in a variety of application areas. The fact that a number of these systems have not been greeted with total enthusiasm by users is a cause for worry, and the reasons for this have themselves become a subject of further study (Markus and Connolly, 1990).

The crucial difference between single user systems and CSCW systems is that the former are designed for use by a single person - if such a product proves a success with a modest percentage of users it is considered a hit. CSCW systems by their nature need to be accepted by a large majority of users in a workgroup or they will fail (Grudin, 1994).

1.2. Supporting Cooperative Work

One aspect of CSCW which is ignored by a lot of products in the groupware category is that CSCW is not supposed to be computer *controlled* cooperative work, but computer *supported* cooperative work (Bannon and Schmidt, 1991). The cooperative aspects of the work need to be decided by the users, and the computer technology is just there to support the cooperation which they require, not to dictate how the cooperation should be achieved. Systems which are too prescriptive in how user groups are expected to work, and which do not take account of the different work practices which groups may adopt, are going to be too inflexible to gain user acceptance in many situations. Users do not want to use CSCW *systems* - they have applications for cooperation, and need CSCW systems to support them.

1.3. Existing User Applications

Another stumbling block in the acceptance of CSCW systems is that many of them insist that users throw away the single user applications which they are accustomed to, and use specially designed multi-user equivalents. This means that users need to have a CSCW equivalent for each of the standard applications which they currently use, and which they wish to use in the new cooperative environment. Not only does this put users off, but it also means a great deal of work for the developers of the CSCW system if they are to provide the range of tools which some users need. They are not always going to be able to emulate the sophisticated commercial applications which are available today.

1.4. Co-ordination and Collaboration

Two of the main objectives in CSCW are those of *collaboration* and *co-ordination* (Ellis et al., 1991). Collaboration in CSCW provides support when users need to work synchronously in a group (i.e. in real-time). Co-ordination supports the organisation of cooperative work among a group of users, both asynchronously and synchronously.

1.5. The Multimedia Toolbox for Cooperative Applications

The Multimedia Toolbox for Cooperative Applications (MMTCA) is an ESPRIT funded project which aims to produce a toolbox to allow end-users to design their own cooperative applications using standard single-user tools. The intention is to integrate tools into the cooperative environment, to allow users to design cooperative procedures, and for MMTCA to provide the appropriate support for the co-ordination and collaboration. The MMTCA consortium has four developer organisations (one of which is the University of Paisley), and three user organisations developing applications using the toolbox.

Co-ordination in MMTCA allows users to design a structured workflow for common administrative procedures - creating a network of tasks, prescribing the creation and use of documents, and allocating resources. Collaboration allows users to co-work on tasks in a distributed environment, allowing shared viewing and editing of selected applications and documents, and control over which users edit documents.

2. REQUIREMENTS

2.1. CSCW Systems and Existing Applications

One of the main requirements which the MMTCA project has is to produce a CSCW system which will actually be put into use. This means allowing users to integrate their standard applications into the system (Greenberg, 1990). This gives them a feeling of familiarity, and reduces the developers' load in trying to emulate a large number of specialist products that already exist and are used in single-user form. It also allows the developers to concentrate on the cooperative aspects of the system, instead of the functional requirements of a particular cooperative process.

2.2. Sharing Information and Structuring Workflow

In cooperative applications, users in a workgroup need to create and store documents, and this information needs to be shared among the workgroup. Therefore workgroup access to shared documents should be catered for, and some sort of access control for when and how documents are used needs to be provided.

To provide consistency for cooperative applications in many areas which may change and evolve over time (possibly as a result of automating the procedure) a generic framework for designing cooperative applications need to be developed. This model should not impose restrictions on how the work process can be designed, although it needs to provide a certain amount of hierarchical structure.

3. TOOLBOX ARCHITECTURE

3.1. The MMTCA Server

MMTCA provides an environment for distributed cooperative work. It uses a client-server model, where user cooperation on client node machines is mediated by a server. This provides central storage for a workgroup's shared information, and distribution of it to user machines. User cooperation is mediated by the server, simplifying communication between users in that they only need to communicate with the server, and the server will distribute the communication to the necessary node machines (MMTCA, 1992). The server can co-ordinate these activities since it stores the workflow information.

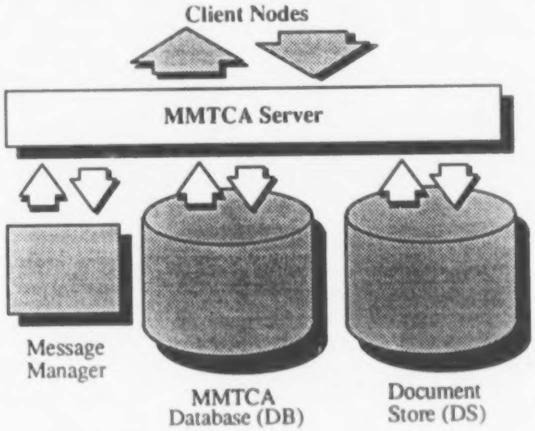


Figure 1. Architecture of the MMTCA server.

660

The MMTCA server provides central storage and control of workflow information. The server has a database to store procedure co-ordination information, administrative information about the users and nodes in the MMTCA system, and procedure instance information about the actual running copies of defined procedures. The server administers to the collaborative needs of users co-working in the system by distributing the information needed for a cooperative session (CS) where users co-work synchronously on the same applications and documents. The server also has the functionality to receive and distribute shared awareness information about the activities of users in the MMTCA system. These forms of information are distributed by the server's Message Manager. The server also provides a document store where documents to be shared by the workgroup are located - the server distributes these to the appropriate users when defined to do so by the workflow which the users have designed.

3.2. MMTCA Client Tools

The client components of the MMTCA system provide tools to design workflow procedures, integrate existing applications into the workflow, create and run instances of the defined procedures, and browse the contents of the database and document store. All the client tools access the MMTCA server through a shared Network Manager layer which provides the services which they require.

3.3. Co-ordination and the PTA Model

The MMTCA architecture provides co-ordination of workgroup activities by allowing users to define procedures for workflow using a client tool, and the procedure definitions are stored by the MMTCA server in the database. MMTCA uses a simple hierarchical *Procedure-Task-Action* (PTA) model for storing workflow on repetitive administrative activities in the database. A procedure can be broken down into tasks, which are functional units of work. A task is composed of a series of actions - the basic unit of work, which may use a number of applications and documents to achieve its goal. Tasks can be defined as *asynchronous* (carried out by a single participant as part of the cooperative procedure) or synchronous (where a group of participants cooperate to complete the task in real-time) - a *Cooperative Session* (CS).

The PTA model defines resources in terms of *documents* and *participants*. Documents can be created in the course of running the procedure, edited during the procedure or just referenced. The procedure definition holds a copy of documents which are defined to exist in the initial procedure state. Participants are defined in the procedure as logical participant roles. These are only mapped to real users when an instance of the procedure definition is created - the mapping can be for the whole procedure, or just on a per-task basis. Resources can be allocated to tasks in a procedure.

The whole combination of participants and documents in an actual instance of a procedure is termed a *metadocument*. When an instance of the procedure is created from the procedure definition, a replica of the initial document set is made for that instance. Real tasks are scheduled for those defined as the starting tasks in the procedure definition.

Procedures can also be defined with generic elements - a logical participant can be designated as being different actual users in different tasks of an instance of a procedure, and a document defined in the procedure may actually be a list containing an arbitrary number of documents which will only be fully specified and stored at run-time.

3.4. Linking Tasks

Task dependencies in the PTA model are defined by the use of *links*. Links define the workflow paths for a procedure. These links can be simple links (one task succeeds another), or more complex links can be created by evaluating conditions and using variables and logic gate links to compute subsequent tasks after a given task has been completed. These low-level link structures are not shown to end-users, but are flexible in the way they can be used to construct higher-level task interconnection structures for workflow control in procedure design notations - e.g. true/false decision boxes and multiple choice option boxes.

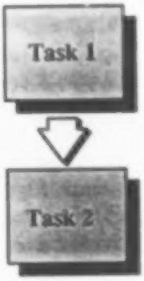


Figure 2a. Simple task links.

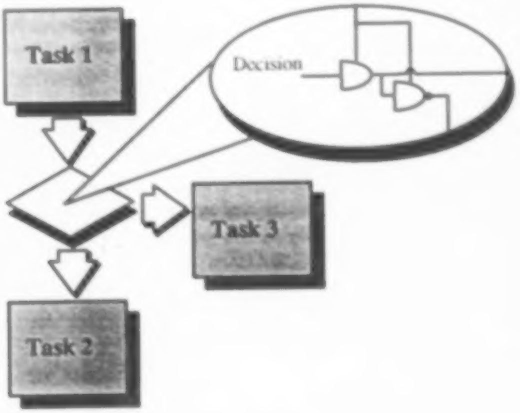


Figure 2b. True/false decision box task links.

The workflow defined in a procedure is not a static process - it can be dynamically altered at run-time in a procedure instance to reflect the way that a particular procedure instance is running. This only requires the procedure design tool to be able to map its notation for linking tasks onto the logic and link structures supported by the PTA model, and for the run-time support for setting conditions to control the workflow to be provided.

3.5. Collaboration Support

Collaboration support in MMTCA is provided by the support for defining synchronous tasks in the form of cooperative sessions. A CS is a task defined in the workflow in the same manner as an asynchronous task except that several participants can be defined to take part in it, a chairman is defined to start the task, and options for the type of "meeting" it will be considered to be are given (e.g. open or closed meeting, the floor policy to be used for mediating who has control at any given time). MMTCA provides run-time support for co-ordinating the starting and ending of the task, the registration of the participants, and the floor control mechanisms.

4. TOOLBOX DESIGN

4.1. Overview

The MMTCA system is implemented in a distributed environment using TCP/IP, with a Unix machine for the MMTCA server (this is currently implemented for Sun, HP and SCO Unix systems). For the server, the standard file system provides the document store (in a directory structure defined by the server process), and the server is gatewayed to an SQL database (currently support is provided for Oracle, Informix and Ingres databases) to provide the MMTCA database facilities. The client side of the system uses PCs running Microsoft Windows 3.1 for the client nodes.

4.2. Server Components

The MMTCA server controls all client communication with the MMTCA database. The file access for client nodes is also controlled by the server, and the database definitions of files stored for procedure definitions and instances are used to control which files are used. The message manager module of the server provides support for the distribution of task control, collaboration and informational messages between client nodes.

4.3. Client Tools

MMTCA provides client tools for designing cooperative procedures, integrating existing applications into the MMTCA environment, creating and running the procedure instances, and browsing the contents of the MMTCA database and document store.

4.4. Designing Procedures

The Visual Editor for Workflow (ViEW) is a tool which allows end-users to graphically design cooperative procedures. The current version of ViEW supports a task-based definition of a procedure - the user defines tasks and the links between them, then assigns participants and actions to the task, and applications and documents for the actions.

4.5. Integrating Existing Applications

The Tool Profile Editor (TPE) is used to store information about existing single-user applications in the MMTCA database. The user can define a template for an application to describe how the single-user customisations can be controlled when the application is to be used in a cooperative session. The customisations need to be standardised across client nodes in a CS, for the purpose of synchronising the client nodes.

In Windows applications, details of customisation of the user interface are usually kept in some form of initialisation file or files. By specifying a common set of files for an application in the tool profile template, the customisations can be standardised for the duration of a CS by distributing the initialisation files to client nodes. Backup copies of the original versions are made, and these are restored after the CS has finished.

4.6. Creating and Running Procedure Instances

The MMTCA Intray is the main run-time application which allows users to create procedure instances, and run tasks from these instances. The intray window shows the tasks which are currently scheduled for the user, and they can select and run a task.

Starting an asynchronous task will start the defined series of actions for the task - downloading the relevant documents from the MMTCA server and starting the applications and loading the documents. When the user has completed the task it can be marked as complete, and the workflow will be recalculated to schedule new tasks on the basis of this. Users can mark tasks as incomplete if they do not finish them within a given session, and return to the task at a later time.

Cooperative sessions (synchronous tasks) involve a more complex setup process of sending out invitations to relevant participant users when the chairman starts the task. Users register for the task, and when a suitable number have registered the task can be started. Documents are downloaded to all participating user nodes, and shared applications started. The chairman of the task becomes the initial floorholder for the CS, and has control of the shared applications. The floor token can be passed among users in a manner defined by the floor policy adopted for the task, so that users can take turns editing documents in the shared applications.

4.7. Browsing the Database and Document Store

The Metadocument Browser (MDB) is a tool which allows users to browse the information in the MMTCA database in a graphical form. The procedure hierarchy information is presented to users for procedure definitions and instances. Where documents are defined in the database description, users can download and inspect the defined documents from the document store.

5. PROGRESS OF WORK

The MMTCA system has been implemented as a series of evolutionary prototypes, which have gradually introduced various elements of the overall architecture. The current prototype implementation is the last to be produced within the lifetime of the project.

The MMTCA consortium contains two user organisations who are developing actual applications which use MMTCA. ISL (Germany) is developing an application for remote fault diagnosis on ships using ISDN networks and satellite connections. Banco del Comercio (Spain) is developing an example application of a banking service for mortgage credit applications in bank branches being administered from a central location. Demonstrations of both applications using the prototype toolbox have been carried out, and it is planned that these applications will be developed further and introduced into real work situations. The example applications have provided feedback for the MMTCA developers in their efforts to provide a usable CSCW system.

A goal of the MMTCA project was the production of the toolbox software itself, but it also provides ideas for future investigation. Problems encountered and concepts developed during the project will be the subject of further research.

Extensive testing of MMTCA in real user situations has not yet been carried out, but feedback from initial user tests, and the evolutionary nature of the prototype design, have already resulted in improvements in some areas. More rigorous evaluation will provide ammunition for further improvements to both the architecture and the implementation of MMTCA.

The University of Paisley is planning to use the principles and architecture from MMTCA in a new project which intends to support cooperation for the early stages of design in a distributed environment. There are also a number of organisations interested in using the MMTCA technology to develop cooperative applications in the workplace, and tracking their take-up and use of the technology will add to the research material which can be used to advance the use of CSCW in real situations.

6. CONCLUSIONS

The MMTCA project has produced a prototype CSCW system which gives users the facilities to design cooperative procedures, and organise information for these procedures. Several of the ideas and features in MMTCA have addressed key areas of study in CSCW. By passing on this technology to end-users the opportunity will be created to study what kind of cooperative applications users see themselves creating, and feedback from them can be gained on what improvements could be made to the system.

It is a little premature to start drawing any conclusions about CSCW issues from the work already done on the MMTCA project - the software itself needs to be extensively used to see how successful the ideas are, and how they affect the way users work together. But by the implementation of a framework for cooperative applications, some of the areas of research have become clearer, and the technology is now available to study them.

7. REFERENCES

Bannon, L., and Schmidt, K., (1991), CSCW: four characters in search of a context, *in:* "Studies in Computer-Supported Cooperative Work: Theory, Practice and Design", J. Bowers and S. Benford, eds., North-Holland, Amsterdam.

Ellis, C., Gibbs, S., and Rein, G., (1991), Groupware: some issues and experiences, *Communications of the ACM*, Vol. 34 (1), pp. 39 58.

Greenberg, S., (1990), Sharing views and interactions with single-user applications, *in:* "Proceedings of the Conference on Office Information Systems", ACM, New York.

Grudin, J., (1994), Groupware and social dynamics: eight challenges for developers, *Communications of the ACM*, Vol. 37 (1), pp. 92 105.

Markus, M., and Connolly, T., (1990), Why CSCW applications fail: problems in the adoption of interdependent work tools, *in:* "Proceedings of Conference on Computer Supported Cooperative Work", ACM, Los Angeles.

MMTCA Consortium, (1992), MMTCA D-2.1 preliminary MMTCA toolbox architecture, ESPRIT Project 6310 Deliverable.

MMTCA Consortium, (1993), MMTCA D-3.31 MMTCA toolbox documentation version 3, ESPRIT Project 6310 Deliverable.

Sproull, L., Kiesler, S., (1991), Computers, networks and work, *Scientific American*, Vol. 265 (3), pp. 84 91.

Some Technical Aspects of ISD

EVALUATION OF THE RESPONSE TIME OF AN INFORMATION SYSTEM WITH DIFFERENT QUERY CLASSES

Aleksander Zgrzywa

Technical University of Wroclaw, Department of Information Systems, 50-370 Wroclaw, Wybrzeże Wyspiańskiego 27, Poland

1. INTRODUCTION

In spite of on-line systems have been used for many years (Zgrzywa, 1980) , new problems in information servicing of users appeared when terminals enabling access to databases found in laboratories, design studies and homes of users. Information servicing specialists help the users to built the "optimal queries" in "classic" selective dissemination of information systems (Kowalski and Zgrzywa, 1982, 1984). Now information retrieval systems have become accessible in the place of work, but the users are deprived of the aid of intermediary searchers. It is the source of essentially new problems that must be solved at various levels of man-computer system communication (Daniłowicz, 1992). Because of big number "not optimal" user's information queries the high performance of the information system is very important also.

Traditionally, information retrieval performance has been assessed using two measures, namely recall and precision. Recall is the proportion of relevant documents (or other information items) in the database that were actually retrieved. Precision is the proportion of documents that were relevant within the set that was retrieved. Recall and precision are useful measures of retrieval effectiveness, but they don't directly address what did the user of the system get out of the search. A supposedly relevant information is no good to the user if he makes no use of it. The third performance measure (without good definition) is utilisation. The effectiveness of the information systems is a controversial topic.

If we are sure that information system has acceptable recall and precision then we may test dynamic performance measures (Baskett et al., 1975). In this paper very important dynamic performance measure - the response time of an information system - is discussed. To obtain the mean response time for the system complicated mathematical methods should be applied (Baccelli and Liu, 1990; Duda and Czachórski, 1987; Walrand, 1988).

This paper deals with investigation of information system (INSPEC) exploited in Technical University of Wrocław. Two other systems: COMPENDEX - COMputerised ENgineering inDEX and BIBKAT - a catalogue information system with database consisting of catalogue descriptions of the books, handbooks, journals, and other documents from University Library were also investigated (Zgrzywa, 1993).

2. ANALYSIS OF INFORMATION QUERIES STRUCTURE

The user queries are regarded as one class, if they have the same structure Examples of the classes are described below (in brackets the processing mean time is given):

1 term$_1$ OR term$_2$ (2.7 s)
e.g. ANALOGUE COMPUTER OR DIGITAL CIRCUIT
2 term$_1$ OR term$_2$ OR term$_3$ (6.1 s)
e.g. SECONDARY EMISSION OR SECONDARY ELECTRON EMISSION OR ELECTRON AFFINITY
3. term$_1$ OR term$_2$ OR term$_3$ OR term$_4$ (11.5 s)
e.g. NEURAL NET OR HYBRID INTEGRATED CIRCUIT OR HYBRID CIRCUIT OR ELECTRONIC CIRCUIT OR CIRCUIT THEORY
4 term$_1$ AND term$_2$ (5.5 s)
e.g. PRODUCT$ AND PETRI NET$
where: $ it is mask on suffix of the term
5. (term$_1$ OR term$_2$) AND term$_3$ (6.0 s)
e.g. (DISTRIBUTED PARAMETER SYSTEM OR CONTROL) AND ESTIMATION
6. (term$_1$ OR term$_2$) AND (term$_3$ OR term$_4$) (6.8 s)
e.g. (SELF-TUNING CONTROL OR DIRECT DIGITAL CONTROL) AND (ADAPTIVE CONTROL OR ON-LINE CONTROL)

The information system under consideration here consists of processor (server - IBM/PC), hard disk, CD-ROM (with INSPEC database) and several terminals (IBM/PC microcomputers). To construct the answer for user query processing time is needed. This processing time consists of processor time and disk times. We say about the service requirements of a user query (e.g. 0.1 seconds for processor, 2.2 seconds for every disk in the system). The service demands were measured using a special program. Then, for a given information system architecture, it was possible to evaluate the mean response time of the system for user queries.

3. EVALUATION OF THE INFORMATION SYSTEM PERFORMANCE USING QUEUING METHODS

A lot of programs for queuing methods have been developed during the past. The most known packages are: IBM's RESQ (Sauer et al., 1983) and French QNAP (Potier, 1984) The evaluation have been provided using AMOK (Czachórski, 1987)

The evaluation has been curried out for users generating queries with different structures and arrival rates. Four query classes: class 1, class 3, class 5, and class 6 were used. For class 1 and class 5 the "thinking" time is Z=15 sec and for class 3 and 6 Z=20 sec. The calculations were provided for N (number of system's users or active terminals) changing from N=4 to N=40, with step equal to 4.

In Figure 1 the response time for class 1 and 5 is plotted against the changing load For every point the number of the queries from the same class is equal to N/2. For growing N the response time of class 1 is augmenting from 2.9 sec to 13.6 sec., and for class 5 from 6.4 sec. to 28.7 sec. These times are not very high because the queries from classes 1 and 5 are very simple.

In Figure 2 the response time for classes 1, 3, 5, and 6 is plotted against the changing load of the system For every point the number of the queries from the same class is equal to N/4. For growing number of queries the response time of class 1 is augmenting from 3.2 to 33.1 sec., for class 3 from 12.5 sec. to 150 sec., for class 5 from 6.8 sec. to 71.5 sec. and for class 6 from 7.9 sec. to 83.4 sec. Despite of the same numbers of users as previously (Figure 1) response times for classes 3 and 6 are very high. It is because the queries from classes 3 and 6 are more complicated than queries from classes 1 and 5. For N=40 the response time for classes 3, 5 and 6 are not acceptable. The number of the users (or in our system terminals) should be not so high or parameters of the information system should be changed after detailed analysis.

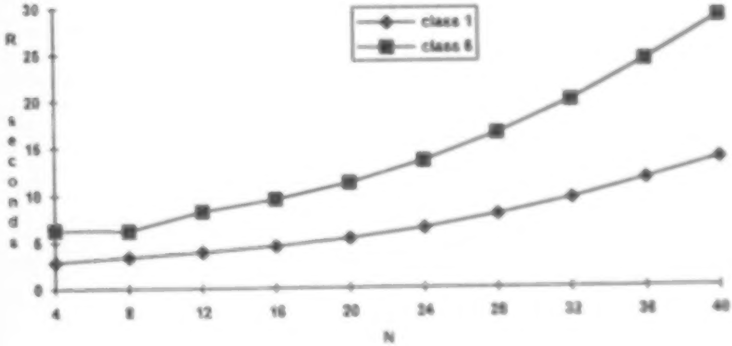


Figure 1. Response time of the information system for users from classes 1 and 5.

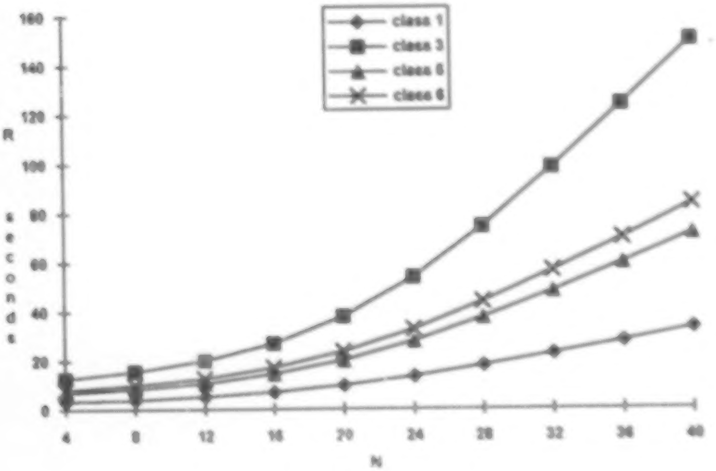


Figure 2. Response time of the information system for users from classes 1, 3, 5 and 6.

4. SUMMARY

In the paper the mean response time for user's queries is investigated because it is very interesting for the users. The users of the terminal information system are paying for their searching time frequently. Hence the response time should be proportional to the number of relevant information items (e.g. documents) obtained as an answer for user's query.

If the number of classes (different structures of queries) is growing the time and space required for calculations are high. For one class the calculation time for microcomputer IBM/PC (even without coprocessor) is less than 1 minute. For several classes calculation time is not greater than a few minutes. If number of classes is greater than 10 only approximate techniques can be used.

The response time (or e.g. throughput of the system) can be evaluate using queuing network methods for every query class or for information system as a whole (global measures). It is possible to obtain characteristics for each element of the system using AMOK package. Such detailed testing is very important if new databases are implemented or a new group of users is giving information queries to the system.

5. REFERENCES

Baccelli, F., Liu, Z., (1990), "On the execution of parallel programs on multiprocessor systems - a queuing theory approach", J.ACM, Vol. 37, No. 2, April, pp. 373-414.

Baskett, F., Chandy, K.M., Muntz, R.R., Palacios, F.G., (1975), "Open, closed, and mixed networks queues with different classes of customers", J.ACM, Vol. 22, No. 2, pp. 248-260.

Czachórski, T., Kowalówka, M., Szczerbiński, Z., Wilk, A., (1987), "Analityczne metody rozwiazywania probabilistycznych modeli systemow komputerowych. Pakiet programowy AMOK", Skrypty uczelniane, No. 1383, Politechnika Śląska, Gliwice, (in Polish).

Daniłowicz, C., (1992), "Models of information retrieval systems with special regard to users' preferences", Scientific Papers of the Main Library and Scientific Information Centre of the Technical University of Wroclaw, Monographs, No. 3, (in Polish).

Duda, A., Czachórski, T., (1987), "Performance evaluation of fork and join synchronisation primitives", Acta Informatica, Vol. 24, pp. 525-553.

Kowalski, K., Zgrzywa, A., (1984), "Evaluation of bibliographic data base operation in an SDI system", Journal of Information Science, Vol. 8, pp. 57-61.

Kowalski, K., Zgrzywa, A., (1982), "Exploitation characteristics of the INSPEC data base in the SDI-system at the Technical University of Wroclaw", Information Processing & Management, Vol. 18, pp. 299-306.

Potier, D., (1984), "New user's introduction to QNAP2", Report Technique No. 40, INRIA, Roquencourt.

Sauer, C., McNair, E.A., (1983), "Simulation of computer communication systems", Prentice-Hall.

Walrand, J., (1988), "An introduction to queuing networks", Prentice-Hall, NY.

Zgrzywa, A., (1980), "Application of computer simulation methods to the investigation of on-line information retrieval systems", In: "New trends in documentation and information", Aslib/FID, London, pp. 145-157.

Zgrzywa, A., (1993), "Some remarks on the information system throughput", In: Proceedings of the ISAT School/IFIP TC 8/WG 8.5 Workshop Bazewicz (edit.), Informatics Library of Universities, Wroclaw, pp. 179-186.

THE CONSTRUCTION OF A PERSONAL COMPUTER SYSTEM WITHOUT A COMMAND-LINE INTERFACE

W. D. Maurer[1]

[1]The George Washington University, Department of Electrical Engineering and Computer Science, Washington, DC 20052, USA

1. INTRODUCTION

Over the long term, it will not be enough for scientists all over the world to continue to use exclusively those personal computer systems which are constructed by the major manufacturers. In each nation, there should be computer scientists who are involved personally in computer systems development and who direct the research of students writing systems programs for personal computers. Our purpose here is to expound the advantages of having this research done on systems which do not use a command line interface.

The worldwide influence of Unix and the availability of very inexpensive used computer systems, most of which are DOS machines, have had as a combined result that many scientists in Eastern Europe have as yet not seen or worked with systems which use a window-icon-mouse interface, rather than a command line interface. Such systems include:

- Microsoft Windows (Microsoft, 1992), available for modern, larger DOS machines (this will not run on 640K DOS machines);
- Windows NT (Microsoft, 1993), a larger version of Windows;
- the Macintosh (Apple Technical Library, 1992), from Apple Computer;
- NeXTstep (NeXT Developer's Library, 1991), available for NeXT and IBM-compatible computers; and
- X Windows and OSF/Motif (Open Software Foundation, 1991), which are extensions of Unix.

For many, the single most important reason to study the window-icon-mouse interface is that DOS machines are themselves being replaced, in the USA, by Windows machines. (Indeed, this is one of the reasons for the availability of so many used DOS machines.) Similarly, X Windows is gaining in popularity among Unix users; while the Macintosh, Amiga, and Atari computers have *always* had window-icon-mouse interfaces.

A window-icon mouse system is always much larger than the corresponding command-line-interface system, just as a compiler is always much larger than the corresponding interpreter. Nevertheless, window-icon-mouse systems are so easy to use, and decrease training costs by so much, that they are well worth the extra initial cost.

2. WHY A WINDOW-ICON-MOUSE SYSTEM?

Our discussion of the design criteria for a window-icon-mouse system starts with the question of why we should have such a system at all. Since the advent of CRT technology, and the display of text and graphics on a television set or a similar computer-specific display device, there have been two main types of system for such computers: the command-line-interface system and the window-icon-mouse system. We will now discuss the advantages and disadvantages of each of these.

A command-line-interface system takes commands from the user from the keyboard. Such systems have existed, historically, in two forms. In the first form, each command is given by a single letter, which is *preceded* by its parameters. Thus X might mean "delete," and to do a Delete All, we might write EX (where E stands for "everything"). Such systems show both the advantages and the disadvantages of conciseness: because there are so few keystrokes, you can work very fast if you know the commands very well, and you can court disaster if you don't. For example, EXIT might mean EX followed by "insert the letter T." (This is not a contrived example; it actually happened on one of the early command-line-interface systems.)

More modern command-line-interface systems such as MS-DOS or Unix (without X Windows) use much more human-readable command lines. The first word on such a line is normally the name of the command; the rest of the line consists of parameters. Some of the command names can look mysterious to the uninitiated (why is "delete" spelled rm, for example?) but these are easy to fix, if you want to fix them, by defining your own command names. (By the way, rm means "remove.")

Why are command-line-interface systems being replaced by window-icon-mouse systems? Why is IBM, after MS-DOS, going to Windows? Why is the Unix community going to X Windows and Motif? Why did SunTools get developed for the Sun (and so on)? The most obvious reasons seem to be these:

- *Command names and parameters are hard to remember.* Most people seem to want a system which tells them what commands there are.
- *It is hard to remember in what context a particular command makes sense.* Most people seem to want a system which gives them this information.
- *Looking for a picture is easier than looking for a word.* If all files appear on the screen as pictures, the one you want can be found quicker than if they appear simply as their names.
- *Pointing to something on the screen with a pointing device is easier than typing its name.* (For one thing, the name can be misspelled.)
- *Scrolling through data that is too large for the screen is easier with a visual metaphor.* This can be seen most easily by comparing a modern scroll bar with (for example) the primitive Unix scrolling system known as `more`.
- *Switching from one context to another is easier if windows are used.*

What are the disadvantages of a window-icon-mouse system?

- It takes more memory than a command-line-interface system. (Someone once tried to squeeze a window-icon-mouse system onto a Commodore 64 by adding 64K of additional dedicated memory. The result is barely usable and lacks many features which are standard on all other such systems.)
- If you know a command name, it is faster to type this than to use the mouse. (For this reason, most window-icon-mouse systems also allow keyboard commands.)
- Blind programmers cannot use such a system at all.

3. SUMMARY OF WINDOW-ICON-MOUSE SYSTEMS

We now summarize window-icon-mouse systems as they work at the present time. For further details, the reader is referred to Maurer (1994).

As computer systems became more powerful, there arose among users the desire to work in several contexts at once. People wanted to work on two word processing documents, two spreadsheets, or even one of each, and to see both of these on the screen at the same time. It is visually limiting if the document on which you are working has to take up the entire screen. This leads to the idea of windows.

A window-icon-mouse system uses a mouse or a trackball or a joystick, collectively known as *pointing devices*. Any pointing device is accompanied by a cursor on the screen.

The form of the cursor icon is immaterial; in fact it seems useful to use several different cursor icons, as a visual indication of what is being pointed to (or of the fact that you temporarily shouldn't be using the mouse, as with the wristwatch icon). The cursor follows the movements of the mouse, and this should *not* be alterable by applications.

There are little pictures, called icons, that represent files, folders (that is, collections of files), and disk volumes. You can move a file by dragging it, and you can delete a file by dragging it to a special icon, such as a trash can.

Commands are given to a window-icon-mouse system through either menus or command-key equivalents. Any command which is not appropriate at the moment appears grayed on its menu. (Turning this graying on and off takes up a tremendous amount of the logic in a typical application.)

Interaction with the user takes place through dialog boxes, which may include push buttons, radio buttons, check boxes, text boxes, and other interactive devices. Interaction with text on the screen takes place by pointing to it, thus clearly indicating "*this* letter in *this* word." Replacement of characters is treated as deletion followed by insertion.

Selection takes place by highlighting (such as inverse video). When something is selected, it can be taken as a command parameter.

Having a window-icon-mouse system at hand even affects the programming languages which you can use. Specifically, you would like to be able to write programs which make use of windows, icons, and the mouse.

For various reasons, windows have to be capable of being moved, re-sized, and scrolled. The most natural way to move a window is to drag it. To change the size and/or shape of a window (called "re-sizing," although this term is a misnomer if "size" means area), it is sufficient to change the position of one corner while leaving the opposite corner fixed. There are many possible ways of scrolling a window. One well-known paint program used a "hand" tool which was dragged to "push" the underlying "paper" around on the screen. This was replaced, in a later version, by what has become the standard tool: the scroll bar with the box inside. It is not clear why this is the best scrolling method; no better one seems to have been found so far.

Having a window-icon-mouse system at hand even affects the programming languages which you can use. Specifically, you would like to be able to write programs which make use of windows, icons, and the mouse. An early programming language of this kind is known as HyperTalk (Winkler, Kamins, and DeVoto, 1994); this bears a vague resemblance to Cobol, in that its statements are "in English." HyperTalk is actually the programming language support for a larger, data-base-oriented system for the Macintosh, known as HyperCard. Another such programming language is Visual Basic (Feldman et al., 1993); this language is available for Microsoft Windows. An extension of the programming language Basic, Visual Basic, like Hypertalk, allows you to specify what you want your program to do when a particular mouse action is taken.

Other aspects of a window-icon-mouse system analyzed by Maurer (1994) include table-driven programs (and their use of special-purpose tables known as resources), the undo function (both single-level and unlimited), fonts and their usage (including font sizes and styles), and multitasking and its uses.

4. FURTHER RESEARCH ON WINDOW-ICON-MOUSE SYSTEMS

The importance of scientific research, in the area of window-icon-mouse systems as in any other area, lies in the exploration of new ideas. Here are some new ideas that seem not to have been tried yet:

- A CRT display could be embedded in a table. If the static electricity could be eliminated, this could be used as a "smart keyboard" with displayed pictures of keys, which could change as the application progresses.
- Mouse buttons could have different colors. "Click green" would be easier to say than "Click the right mouse button."

- A mouse could have *no* buttons; the space bar on the keyboard could be used as a button, for example.
- Special-purpose icons, such as the trash can to represent deletion, are not as much used as they could be. Besides a copy machine to represent duplication, one could conceive of a printer to represent printing; you should be able to drag a file to any of these. There are other possible special-purpose icons that would not involve the dragging of a file. These would include a scissors, for cutting; a bottle of paste, for pasting; a keyboard, for invoking a word-processing system; a can of paint, for invoking a painting program; and the like.
- The buffer in which individual pixels are kept is usually surprisingly primitive — a double array of pixels, each of which is either a single bit (for a black-and-white system) or a small number of bits (for a color system). The problem with such a buffer is that it takes a long time to change, when even simple changes are called for (such as dragging a window). A more sophisticated buffer would involve pointers to several double arrays of this kind; a window could be moved by changing only a few pointers.
- Traditionally, there are several menus, and their names appear on a *menu bar* at the top of the screen, but neither of these decisions are necessarily better than their alternatives. For example, why are there several menus? For no good reason, it would seem, particularly when you know what command you want, and have to search several menus to find it. There could be a single place to click whenever you want to see all the commands, and then these would fill the screen, allowing you to select one with the mouse.
- Why doesn't the text of a radio button, together with the button itself, have a line drawn around it, like a push button? (This would alert the user that you don't have to click on the button; you can click on the text — something that is not obvious.)
- Why isn't there a true *check* box, using a check mark (✓) rather than simply ×?
- Why aren't there standard system routines which know about groups of radio buttons? Why, indeed, are radio buttons still used, when pop-up menus take up less space?
- We should go through the functions of an editor such as *vi* and try to find visual metaphors for all of them. That job, if indeed it was ever started, was apparently never finished. Consider, for example, the exchange function which swaps two letters (turning ADN into AND, if we swap the D and the N). It would be easy to point to the space between the D and the N, and then click command-E for "exchange."
- There is a general implementation of Undo that sounds very easy: every time you change a variable, you push its location and its old contents. When you start a function, you push zero, which cannot be a legal location. When you undo, you pop a location and its contents, and you check the location. If it is zero, you stop; otherwise, you put the contents back in the location, and loop back to pop two further quantities. The scheme applies without change to unlimited Undo. It has to be extended when you are changing files, rather than variables; you have to keep old versions of files somehow.

5. REFERENCES

Apple Technical Library, (1992),"Inside Macintosh: Overview", Addison-Wesley, Reading, MA

Feldman, P., et al., (1993), "Using Visual Basic 3", Que Corp., Indianapolis.

Maurer, W. D., (1994), "Personal computer systems without command-line interfaces", Report GWU-IIST-94-12, Institute for Information Science and Technology, George Washington University, Washington.

Microsoft, (1992), "Microsoft Windows 3.1 Programmer's Reference", Microsoft Press, Redmond, WA.

Microsoft, (1993), "Microsoft Win32™ Programmer's Reference", Microsoft Press, Redmond, WA.

NeXT Developer's Library, (1991), "NeXTstep Reference", Addison-Wesley, Reading, MA.

Open Software Foundation, (1991), "OSF/Motif Programmer's Reference", Prentice-Hall, Englewood Cliffs, NJ.

Winkler, D., S. Kamins, and J. DeVoto, (1994), "Hypertalk 2.2 — The Book", Random House, New York.

COMPUTER SIMULATION MODEL FOR COST-BENEFIT ANALYSIS

Miro Gradišar
University of Maribor, School of Organisational Sciences, 64000 Kranj, Prešernova 11

1. INTRODUCTION

DSS (Decision support systems) and EIS (Executive Information Systems) tools enable efficient use of methods for planning and analysis of costs and revenues for the needs of decision making and management. Despite the fact that such tools are relatively expensive from the point of software view, they are nevertheless gaining popularity in Slovenia. This assumption is also confirmed by a study of key cases at managing informatics in Slovene companies in 1993 (Zupančič et al., 1993).

Cost-benefit analysis is enabled by several methods. In business systems we are most often faced with the direct-costing method (Melavc, 1989). Our purpose, therefore, is not to elaborate a computer simulation model which would enable the use of one particular method only, but to design it in such a way that it could be used, by setting parameters, for different purposes which would represent implementation of various methods.

2. METHOD

Although there are many relevant papers describing the method of model building and methodology of its application (Forrester, 1961; Gordon, 1969; Coyle, 1977; Birta, 1990; Kljajić et al., 1991), there are no commonly accepted rules of their validation that would be a guarantee of model quality and its acceptance. When developing and using computer simulation models for business decision support we are acquainted with the basic issue of validation (Banks, 1990; Kljajić et al., 1990). The bigger the model, the more difficult is its validation. (Ross et al., 1988), expresses the opinion that the use of complex models is too risky. On the other hand, the too simple models for cost-benefit simulation might be inaccurate. We will try to resolve this conflict by using a model design which will enable the link with cost accounting for a specific past period. The cost accounting itself has also to be organised in a way to enable such a link. This will assure us of automatic updating of the model parameters which would thus correspond to the state for this past period. Each individual simulation project wouldn't have to be implemented in all phases (Gordon, 1969). We would only have to implement the last phase and use the already tried and tested model.

Preparation of data for experiments would thus contain only the anticipated deviations from the momentary values in future period for which the simulation is being executed. However, this could not be always possible. When simulating new projects, which have no history yet, validation phase should also be implemented.

3. MODEL

Work with the model is run via menus having eight hierarchic levels with three basic branches:

- setting internal parameters
- setting external parameters
- implementation of simulation and results listing

4. RESULTS

Here are, as an example, results of the simulation of a simple hypothetical case using the following data:

- internal parameters:
  - common parameters:
    - time step: 1 month; number of steps 12
    - quantity step: 91 pieces; number of steps: 11; initial value: 1
    - anticipated profit: 800 money units per month
    - fiscal elements: 40%
  - cost elements:

    Cost elements in money units are displayed in table 1.

Table 1. Cost elements

cipher of el.	description	fixed-dir.	fixed-ind.	var.-dir.	var.-ind.
100000000001	depreciation	324	45	0	0
100000000002	material costs	0	0	1.350	0
100000000003	labour costs	0	0	1.070	0
100000000004	maintenance	70.1	30.5	0	0
100000000005	energy	0	0	0.200	0
100000000006	other costs	68.5	173.4	0	0

- external parameters:

  External parameters are set in the way that demand is fluctuating between 300 and 1000 of units and selling price is fluctuating between

10.30 and 11.50 of money units.

- simulation results:

  Changing cost price per unit in dependence with quantities is displayed in figure 1.

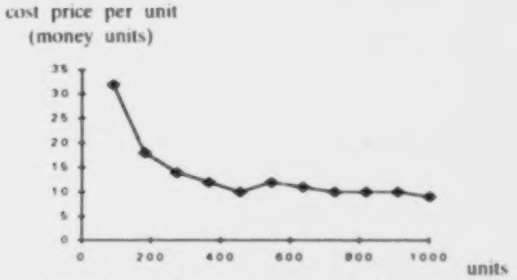


Figure 1. Dependence of cost price per unit from produced quantities

Trend of selling price and sold quantities in dependence with time is displayed in figure 2.

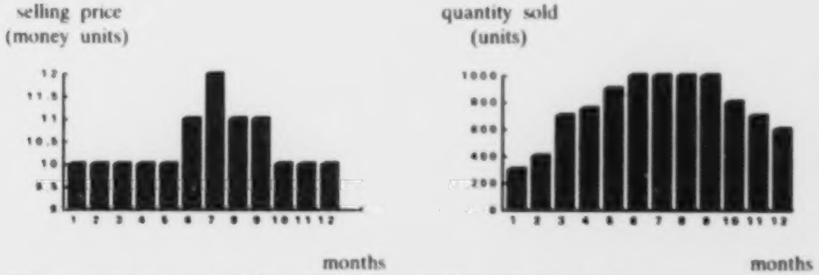


Figure 2. Dependence of selling prices and quantities sold from time

Profit and cumulative profit for the period of one year are displayed in figure 3.

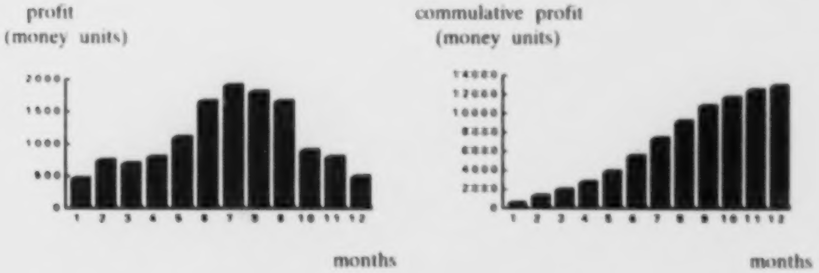


Figure 3. Profit and cumulative profit

After data entry the duration of simulation takes a few minutes only. In this way, by means of what-if analysis, many alternatives can be evaluated in short time, enabling thus precious information for business decision.

Let's take it that we want to decide about the high of selling price, which would, at the anticipated sold quantities on individual market segments, bring the maximum profit. The simulation results show that the profit in our case is already higher than the planned one. By further testing we might find even more favourable result.

5. CONCLUSION

The demonstrated model could be defined by the following attributes: computer, quantitative, deterministic, multi-criteria and heuristic. It is designed as to consider optional grade of detail, therefore it can occur when it is more comprehensive and accurate as a part of DSS and a part of EIS when it is less comprehensive and prepared in advance in aggregated and generalised form.

6. REFERENCES

Banks J. (1990), Verifying and validating complex simulation models by analogy, Simulation, Vol.54, No.1

Birta L. G. (1990), A database support environment for numerical experimentation, Simulation, Vol.54, No.4

Coyle R. G. (1977), "Management System Dinamics", John Wiley & Sons

Forrester J. W. (1961), "Industrial Dynamics", MIT Press, Cambridge, Mass.

Gordon G. (1969), "System Simulation", Prentice-Hall

Kljajić M., Leskovar R., Gradišar M. (1991), Management system simulation model and its validation, Proceedings of the 13th International Conference Information Tchnology Interface, Cavtat

Kljajić M., Leskovar R., Gradišar M. (1990), Validation of management simulation model and its parameter sensitivity analysis, IASTED, Applied Simulation and Modelling, M. H. Mamza (ed.), Acta Press, Anaheim-Calgary-Zurich

Melavc D. (1989), "Računovodstvo danes", Moderna organizacija, Kranj

Ross S.-C. et al., (1988), "Developing and Using Decision Support Applications", West Publishing Company, St. Paul

Sprague R. H. McNurlin B. C. (1993), "Information Systems Management in Practice", Prentice-Hall, Englewood Cliffs

Zupančič J., Dekleva S. (1993), Key issues in information system management in companies in Slovenia, XII. meeting of work organisers, Portorož, pp. 163-167

DATABASE STRUCTURE AND DISK SPACE REQUIREMENTS OF COMPUTER AIDED MANAGEMENT SYSTEMS

Marek Miłosz[1] and Elzbieta Miłosz[2]

[1] Department of Computer Science
[2] Department of Management
Lublin Technical University, 20-618 Lublin, ul Nadbystrzycka 36B, Poland

1. INTRODUCTION

Most Computer Aided Management (CAM) systems in Polish middle-size companies are realized on technology, which uses Relational DataBase Management System (RDBMS) of dBase type (dBase, Clipper, FoxPro, etc.).

Theese systems are used in local area network consisting at PC type microcomputers and network operating system Novell NetWare. This specific technical and program structure of CAM systems results from the history of Polish computer science development, political and economic conditions which had been a reality in Poland until quite lately. More advanced software products (including modern RDBMS with 4GL languages) were in reality inaccessible, because of COCOM limitations.

At present time, more and more Information Systems (IS), including CAM systems, are created in such RDBMS as: GUPTA, INFORMIX, INGRES, ORACLE or PROGRESS. In case of large IS, the use effectiveness of above-mentioned tolls is unquestionable and experimentally confirmed. IS, based on RDBMS of dBase type, failed in banks, big companies or government and local administration. Programmers' expertness did not help -- systems did not reach planned effectiveness (after database increasing), systems response time was past all permissible limits, problems with database management and system maintenance arose. The important problem is connection of almost all dBase type systems with MS-DOS platform, missing client-server architecture and impossibility of distributed data processing.

After winning large IS market, advanced RDBMS are more and more often applied used in middle-size companies. This process involves overcoming many barriers. One of them is notion, that advanced RDBMS are very expanded and need big resources, most of all require big mass store.

In one of the mentioned advanced RDBMS, database is not a simple file set anymore, as it is in case of such systems as Clipper. Theese database has more complicated structure, which besides definitions at each table and index, contains also quite a few additional elements. They are integrity rules, validation criterion definitions, labels, elements of data security, tables' relationships and service procedures of events (triggers). Such database dictionaries need large disk space, even when they do not contain any date. Theese elementary requirements are not small. For example, entirely empty database dictionary of PROGRESS system v. 6.2 for DOS takes about 39 KB, while small CAM system dictionary of this RDBMS requires over 100 KB.

Advanced RDBM systems store data in variable length records, what in some cases also causes increase of disk space requirements.

2. STRUCTURE AND CONTENT OF CAM SYSTEMS' DATABASES

In order to define a real database structure and disk space requirements in CAM system, some account systems' files with system data were analyzed. Thirteen systems coming from various producers were tested and subjected to statistic handling.
Theese systems were operated in middle-size companies (employed 200-500 workers) of various activity areas and they contained constant data (i.e. dictionary) and account records of one month.

Statistics' data was averaged taking into account weight factor, for which the record number in file was taken.
Account systems' databases consisted of 75 files (tables) and 95 index files in average. Mean length of index key arrangement 15.5 symbols (characters).
Disk space taken by theese databases fluctuated from 3 to 10 MB. Practically in every system there were 2-3 very large files, which with their index files took 80-90% of database disk space. Database files consisted of 2-24 fields (mean 13.8). Distribution histogram of field number on database file regarding record number is shown on figure 1.
Table 1 pictures structure of file records, that is an every type fields' percentage on database.

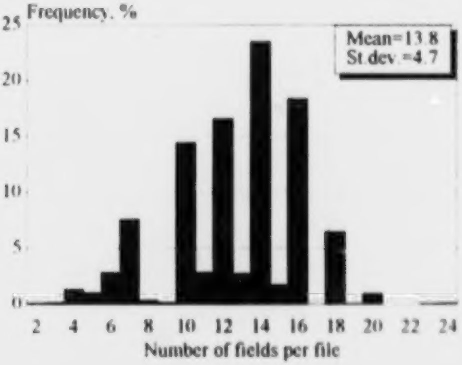


Figure 1. Frequency distribution of number of fields per file on database.

Besides database record structure, also statistic distribution of declared and real used lengths of fields were investigated.

Some survey results are shown on table 2. Both numeric and character fields are characterized by low contents. Contents index of theese fields is equal 39% and 35% as

Table 1. Structure of account system databases' file records

Field type	Percentage of fields number	Percentage of fields number with regard to length declared	Percentage of fields number with regard to real contents
Character	45.6	47.8	45.1
Numeric	40.9	41.0	32.5
Date	12.8	11.1	21.0
Logical	0.7	0.1	1.4

appropriate. That means, that almost 40% of numeric fields contain zeros and 35% character fields do not contain any information. At the same time 7.5% of numeric fields and over 21% of character fields are completely filled. Theese fields are usually used for identification (codes, symbols, indexes, documents' numbers, etc.). Long fields (both numeric and character) are very seldom completely filled.

Table 2. Statistical distribution parameters of database record structure

Parameter fields	All date		Without empty	
	Mean	St. deviation	Mean	St. deviation
Numeric fields storage	9.2	5.0	-	-
Numeric fields contents	2.9	3.7	4.8	3.5
Index of numeric fields contents, %	35.7	33.7	58.9	21.7
Character fields storage	9.6	9.2	-	-
Character fields contents	3.8	5.1	5.8	5.2
Index of character fields contents, %	44.6	38.9	68.6	26.2

Databases of dBase type characterize big differences between declared and real lengths of fields. This approach to database design in dBase type systems results from a simple thing: length of database record is constant. Thus lengths must be declared with appropriate reserve, just in case. Because of constant length of record dBase type databases are very rare. RDBMS of PROGRESS type do not have this disadvantage. Variable length of record enables reduction of disk space taken by database.

3. COMPARING OF DISK SPACE REQUIREMENTS ON DATABASE IN CLIPPER AND PROGRESS SYSTEMS

Comparison of disk space requirement with CAM systems' databases was investigated for two RDBMS: Clipper and PROGRESS. They are the systems (each in their own type), which are most often used at present in Poland for CAM systems development. Experiment consisted in generation of tested database with various record numbers and size comparison of disk space requires for its storing in both RDBMS.

Tested databases were composed of a file with length record equal 146 characters. The record contained 14 fields; 7 of character type, 5 of numeric type and 2 of data type in it. Fields' lengths and their contents were related to statistics' information obtained as a result of studies on existing CAM systems. Fields of data type were completely filled. Index file with 10-character key was included into a database. Completely filled character field was then an index key.
Total size of all database files was taken for a database created by clipper system file lengths amounted: .dbf (database file) and .ntx (index file), and for PROGRESS: .db (dictionary, data and index), .bi (before image file, empty in this case) and .lg (loginscript file).

Investigation results of tested databases' size are shown on figure 2.
For a small number of records, database created by PROGRESS is bigger than database of dBase type, but in case of 1000 records they are becoming equal. Dependence of database size on a record number for both RDBMS is practically linear (figure 2).

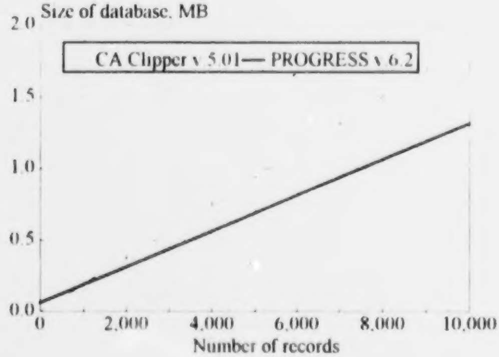


Figure 2. Dependence of disk space size on a record number on tested databases

In case of IS database in size of 3-10 MB disk space requirements of PROGRESS database are 1-3,5 MB smaller than database of dBase type. Taking into account the fact, that in middle size companies there are 3-6 various CAM systems used, each in a different field of activity, disk space savings can be remarkable. The savings regard not only hard disk space but also data carriers archival copies.

4. SUMMARY

Use of technologically advanced RDBMS for creating of CAM systems in middle-size companies, besides other advantages, enables reduction of demand on mass store (on hard disk, archive media, for mirroring, duplexing and others). The savings result from different organization of data storage on database and real data structure of CAM systems.
Obviously those are not the only advantages resulting from use of technologically advanced RDBMS and they also should not be the only criterion of system choice.

A COMPARATIVE ANALYSIS OF SIX MANUFACTURING SIMULATORS

Vlatka Hlupic

Brunel University, Computer Science Department, Uxbridge UB8 3PH, Middlesex, United Kingdom

1. INTRODUCTION

Advanced manufacturing systems are being increasingly used in many industries in order to improve productivity and to reduce costs. The complexity and dynamic behaviour of such systems are the main reasons for the use of simulation modelling as a modelling tool for facilitating their design and assessing operating strategies.

A growing popularity of simulation is reflected by a growth in the number of simulation software products in the software market. This paper presents an evaluation and comparison of six widely used data driven manufacturing simulators. The evaluation is not performed in order to discover which is 'the best' simulator, because such a term does not exist in the context of simulation software. The main reason for this is a constant revising of existing software and the release of new software products.

Subsequently of a review of background research material, the manufacturing simulators to be evaluated are briefly introduced, as well as an evaluation framework used for their evaluation and comparison. On the basis of the evaluation, a method of rating simulators is proposed, and the grades are given to easch simulator. The conclusions outline the main findings derived in this research.

2. BACKGROUND RESEARCH MATERIAL

The research presented in this paper has been initiated by a review of previous studies on the evaluation and comparison of simulation software tools. Though there are many studies that describe the use of particular simulation packages or languages (Fan and Sackett (1988), Taraman (1986) etc.), relatively few comparative assessments were found.

Carrie (1988) presents features of GASP, EXPRESS, GENETIK, WITNESS and MAST, but without an extensive comparison. Banks *et al.* (1991) evaluate SIMFACTORY II.5, XCELL+, WITNESS and ProModelPC by modelling two manufacturing systems. A simulation software survey and evaluation is carried out by Law and Haider (1989) on the basis of information provided by vendors. Both simulation languages and simulators such as FACTOR, MAST, WITNESS, XCELL+ and SIMFACTORY II.5 are included in this study. Grant and Weiner (1986) analyze simulation software products such as BEAM, Cinema, PCModel, SEE WHY and SIMFACTORY II.5, on the basis of information provided by the vendors.

An analysis of the above studies in simulation software evaluation and comparison shows that several evaluation studies are based on information provided by vendors, and lack any criticism. It seems likely that many authors did not have an opportunity to test all the software tools considered and use them for developing complex models of real systems. Though some of the evaluation studies consider WITNESS, SIMFACTORY, XCELL+ and ProModelPC, none of these evaluations and comparisons is comprehensive nor they include evaluation of INSTRATA and AUTOMOD II. For these reasons, this research aims to produce a more extensive and critical evaluation of manufacturing simulators, based on a detailed case study related to a real manufacturing system.

3. EVALUATION OF MANUFACTURING SIMULATORS

This research provides an evaluation of several manufacturing simulators: WITNESS, SIMFACTORY II.5, ProModelPC (DOS version), INSTRATA, AUTOMOD II and XCELL+. Although all of these simulators are data-driven, visual, interactive, and manufacturing oriented, there are many differences between them. A detailed description of these simulators is provided in Hlupic (1993).

Evaluation has been carried out on the basis of a case study, which relates to an automated system for electrostatic powder coating of metal components in an electronics company, operating in the United Kingdom (Hlupic and Paul, 1993a). Several simulation models of this systems were developed, using each simulator under consideration. Due to the main characteristics of the system being modelled, a variety of features had to be considered during the modelling phase, and this provided a good basis for the evaluation of the simulators.

The evaluation framework described in Hlupic and Paul (1993b) formed a basis for evaluation. This framework comprises more than 330 criteria grouped into 17 groups. These groups were used as the base for rating the simulators. Such an approach was taken because it was assumed that it is more convenient and useful to assess the general performance of each software tool with regard to a particular group of criteria, rather than to evaluate every single criteria.

4. COMPARISON AND RATING OF THE EVALUATED SIMULATORS

This section provides a comparison and rating of the evaluated simulators. Information presented here is originated from the evaluation of these simulators as well as from the overall impressions and experience gained through learning and using these simulators. Whilst this paper merely provides the grades given to simulators according to their performance in terms of various groups of criteria, a detailed analysis and justification of these grades is given in Hlupic (1993).

A rating of the evaluated simulators has been established for the purpose of their comparison. This rating should be considered as a relative measure of quality of these simulators from the perspective of groups of criteria, rather than as an absolute value.

Table 1 shows a proposed rating for the simulators being evaluated, in terms of the general quality of features within particular groups of criteria. The rating interval used in this assessment is similar to the one proposed by Ekere and Hannam (1989). The general quality of simulators with respect to particular groups of criteria is rated from 1 to 10, where 1 represents very poor quality or absence of the features within particular groups of criteria, whilst grade 10 represents excellent quality. Accordingly, it is proposed that 5 is taken to be a 'nominal acceptance level', or NAL for short. The grades for a certain group of criteria that

684

are above the NAL indicate that a package is performing adequately, whereas those below signify the opposite. Whilst the NAL is clearly subjective, it does provide a level against which the relative performance of a package can be measured and reflected on. Since evaluation cannot be entirely objective, this qualitative measure of performance, the NAL, does provide a relative measure. However, clearly any particular grade is merely a 'qualitative' number, and the rules of arithmetic can only be applied with caution and with caveats, if at all.

Table 1. Comparison of evaluated simulators in terms of groups of criteria

SIMULATORS GROUPS OF CRITERIA	WITNESS	SIMFACT.	INSTRATA	AUTOMOD II	XCELL +	ProModel PC (DOS version)
General Features	8	8	7	8	7	7
Visual Aspects	8	7	7	8	5	6
Coding Aspects	7	5	6	7	1	6
Efficiency	8	7	6	7	6	7
Modelling Assistance	8	7	7	8	7	6
Testability	8	7	6	8	6	5
Software Compatibility	6	7	6	6	6	7
Input/Output	8	7	7	7	6	7
Experimentation Facilities	7	8	7	7	6	8
Statistical Facilities	7	8	7	7	5	7
User Support	8	8	7	7	7	7
Financial and Technical Features	4	6	5	5	7	8
Pedigree	9	8	8	8	8	8
General Manufacturing Modelling Features	8	8	8	8	6	7
Physical Elements	8	8	8	7	6	7
Scheduling Features	8	7	7	7	5	7
Manufacturing Performance	8	7	7	7	6	7

5. SUMMARY AND CONCLUSIONS

This paper provides an evaluation of several manufacturing simulators. A detailed critical evaluation including positive and negative features of each simulator (derived from the perspective of groups of criteria) is given in Hlupic (1993). During the evaluation not every single criteria within each group was examined, because the aim was to generally perceive basic features of each simulator. Specific features are probably going to change and be added to with new releases of the simulators under consideration.

A comparative analysis of the evaluated simulators is provided. The general quality of each group of criteria was ranked for each simulator. This revealed that although all simulators belong to the same type of simulation software, there is a variety of differences between them. In addition, none of the simulators satisfies all criteria, and none is equally good for all purposes. Although some simulators are more comprehensive and flexible than others, a simulator that can suit any manufacturing problem does not exist. At the same time those simulators that are more robust and adaptable are usually more expensive and difficult to learn and use properly. This confirms the statement that "the less work required of the user the more must be done by the package itself, which increases its complexity, size, cost and execution times" (Carrie, 1988).

The fact that the selection of a piece of simulation software is a matter of compromise between many factors is substantiated by this research. One of the most important factors that determines which software is more suitable than others is its intended purposes. Other factors to consider are financial constraints and subjective factors such as individual preferences and previous experience in using simulation software.

REFERENCES

AUTOMOD Users Manual, (1993), Autosimulations inc, Utah, USA.

Banks J., Aviles E., McLaughlin J.R., Yuan R.C., (1991), "The Simulator:New Member of the Simulation Family", *Interfaces*, **21**(1), March-April, pp.76-86.

Carrie A., (1988), *Simulation of Manufacturing Systems*, (Wiley, Great Britain).

Conway R., Maxwell W.,McClain J., Worona S., (1990), *User's Guide to XCELL+*, (The Scientific Press, San Francisco).

Ekere N.N., Hannam R.G., 1989, "An evaluation of approaches to modelling and simulating manufacturing systems", *International Journal of Production Research*, **27**(4), pp.599-611.

Fan I.S., Sackett P.J., (1988), "A PROLOG Simulator for interactive flexible manufacturing systems control", *Simulation*, **50**(6), pp.239-247.

Grant J.W. and Weiner S.A., (1986), "Factors To Consider In Choosing A Graphical Animated Simulation System", *Industrial Engineer*, **31**, August 1986, pp.37-40, 65-68.

Hlupic V., (1993), *Simulation Modelling Software Approaches to Manufacturing Problems*, Unpublished Ph.D. Thesis, The London School of Economics, The University of London.

Hlupic V. and Paul R.J., (1993a), "Simulating an Automated Paint Shop in the Electronics Industry", forthcoming in *Simulation Practice and Theory*.

Hlupic V. and Paul R.J., (1993b), "Evaluation Criteria for Manufacturing Simulators", submitted to *Production planning and control*.

Hlupic V. and Paul R.J., (1993c), "Guidelines for the Selection of Manufacturing Simulation Software", submitted to *Information Processing and Management*.

INSTRATA Manual, (1992), Insight Logistics Ltd, Oxon.

Kochhar A.K., Ma X., (1989), "Use of Computer Simulation:Aids for Solving Production Management Problems", *Proceedings of the 3rd European Simulation Congress*, Edinburgh, 1989, pp.516-522.

Law A.M. and Haider S.W., (1989), "Selecting Simulation Software for Manufacturing Applications: Practical Guidelines & Software Survey", *Industrial Engineer*, **34**, May 1989, pp.33-46.

ProModelPC User Manual, (1991), Version 5.0, (Production Modelling Corporation, Orem).

SIMFACTORY II.5 Reference Manual and User's Guide, (1990), (San Diego, California, CACI Products Company).

Taraman S.R., (1986), "An Interactive Graphic Simulation Model for Scheduling the Factory of the Future", *AUTOFACT'86 Conference Proceedings*, Detroit, pp. 4-31,4-3814.

WITNESS User Manual, 1991, (AT&T ISTEL Ltd., Redditch).

Work in Progress

INTELLIGENT AND COOPERATIVE CASE TOOL KIT FOR DEVELOPMENT OF INFORMATION SYSTEMS

Antonio Guevara-Plaza, Andrés Aguayo-Maldonado

Dept. Lenguajes y Ciencias de la Computación, Facultad de Informática, Universidad de Málaga,
Plaza del Ejido, 29013 Málaga, Spain, e-mail: <guevara, aguayo>@ctima.uma.es

1. INTRODUCTION

Beyond vendors' hype and theorists' clear-cut formalizations, users and professionals of Information Technology (IT) have realized that many problems affect the actual production of software. It is the so-called *software crisis*. This crisis is specially perceived in organizations that present any of the following features:

- complex hierarchy or structure (for example, organizations which can be described better in terms of a network or matrix structure than a pyramidal hierarchy)
- time-dependent data flow (for example, short-lived sources of data, or daily or seasonal variations in the structure of the dataflow)
- distributed organizational control

IT must address these problems in order to become a useful tool for big organizations. It is not an academic exercise: actual installation of IS in such organizations can cause not even a slight improvement in the productivity, but noticeable delays and slowing-ups in the behaviour of the whole organization.

In the following section the problems arising in conventional Information Systems (IS) are briefly exposed. Then an alternative for the design is proposed, and finally a part of an implemented cooperative CASE tool is described.

1.1. Some issues in conventional IT

The following four sets of problems take place in sequence when planning an automated Information System:

- How must be IT inserted into the organization?
- How can be the applications expanded?
- How can be the coordination achieved?
- Is it possible to describe the dependencies between the organization, IT and IS?

The problems in the first stage are conventionally solved by creating a new department inside the organization or by consulting an external organization. In any case, the management usually delegates the IS tasks in a IS coordinator that must control the development and implementation of the system. Users are outside the IS, and their only interaction with the developers is the definition of needs via the coordinator.

In the second stage, applications must be expanded in order to satisfy the increasing needs of the users. Due to communication problems between users and IT professionals, the IS department or the external consultant fails to solve the proposed needs, and implements, say, only 70% of the requested task. The detected disfunctionalities are fedback to the IS department, and so on. By means of a stressing sequence of such cycles, the applications are finally expanded without achieving a perfect accommodation to the user needs.

And it is in this scenario where the third kind of problems appears. The solution of these problems is beyond the capability of any isolated department, so the management is forced to formulate a set of policies to assign priorities to the different users and applications. These assumption of a new role frustrates in a certain way the expectations of IS department, that sees how its past power is diminished. This new problem must be addressed by means of an accomplished relationship between all the actors involved in the IS design and use: IS professionals, management and users. In this way, the achievement of competitive advantages is the main goal for the IS, and IT is at service of the organization in a rational way.

1.2. An alternative paradigm

The precedent informal discussion shows that the main problems in IS development are addressed only in the final stages of the life cycle of the system. An alternative paradigm has evolved based on the use of CASE tools in an cooperative intelligent environment. CASE tools have a long history inside IT. They are used in an increasing number of organizations, with verifiable effects on the productivity of system analysts and programmers. Their range is very broad, from the cheapest tools running on PC's to the more complex and expensive ones demanding mainframe hardware. These tools usually display and generate some kind of diagrams, and sometimes also COBOL or other programming languages code.

Software reusability is a goal pursued by both conventional CASE tools and AI programmers (Cuena, 1992) (Steels, 1992). The final aim is to allow non-programmers to construct their own applications, by configuring and assembling preexisting pieces of code. Also programs developers could reuse their own code, or code of people in the same organization. The favourite conceptual tool that AI researchers mention in this context is the distinction between *symbolic level programming* and *knowledge-level programming* (Newell, 1982). Pieces of code lay at the symbolic level; final users and meta-programmers reason in an upper layer, that of the "knowledge", handling complex structures matching with symbolic level pieces. In any case, current CASE tools lacks of real cooperativity. The system is designed and implemented in a conventional, hierarchical way. So, users have little direct contact with the definition of the system, and different views must be harmonized by the IS technicians' power and authority.

Another trend in IT stresses the importance and utility of desegregation, from distributed computing at the hardware level to distributed databases at the data level. Extrapolating this trend to describe human interactions, we have the cooperative work, and by using IT as a tool to allow new kinds of cooperative work, we arrive to CSCW.

This feature must be reflected in the organizational IS at least in two ways.

Type A cooperativity: the IS must provide easy ways to communicate and share information between many agents, sometimes at different geographical and organizational locations.

Type B cooperativity: the IS definition must be the result of the collaboration of several users. Type A cooperativity could be implemented as a simple mechanism of physical networking. Nevertheless, a lot of theoretical and practical effort has been invested in order to develop more or less intelligent networks supporting cooperative work (Shepherd et al, 1990).

In the rest of the paper, we focus onto type-B cooperativity, that demands a new paradigm for the development of the IS (Guevara, 1992). This new paradigm can be summarized as follows:

From the beginning of the life cycle, a set of users are selected in order to develop the system. The users are trained by the IS department in the use of a CASE tool based upon data dictionaries (DD), data flow diagrams (DFD) and entity/relationship diagrams (ERD). Each user defines his own data by means of a conventional CASE tool. This definition enables the cooperative system to create a local DD for each user, and a global DD for the organization. On the other hand, each user designs his own level diagrams. Then another CASE tool collects all the diagrams and produces the context levels and the first levels, representing the global situation of the organization and the distribution of the information between the users. So, the tool allows a first top-down definition and then accomplishes a second automated bottom-down design.

The user also decomposes his processes until the lowest level -from his own point of view- is reached. At this level the user must write a description of the elementary process -ideally, in a subset of a natural language as flexible as possible. A CASE tool will generate from these data the structure diagrams and database structure.

So, the whole definition and design of the IS is viewed as a cooperative task carried out mainly by the users, who interact with a CASE tool that allows free cooperation between them. Strictly speaking, we are not faced with a full cooperative architecture, since continuos contact between agents is not required. Here, the cooperative flavour is given by the integration of the work from autonomous agents and by the resolution of the conflicts arising in the integration process, defined in a partially cooperative architecture.

In the following section we will describe the prototype of a tool developed by our team that implements part of the first issue of the above exposed.

2. AN EXAMPLE: THE COOPERATIVE DATA DICTIONARY

It is obvious that the systematic collection of a huge amount of data is not an easy task when it is attempted to reflect the whole functionality of a big organization. Many people are involved in this task, and an intelligent tool allowing the cooperative work would be very useful for great or distributed organizations. Such a tool will favour group work and maximize the efficiency of invested efforts by shortening develop time, and will yield a better DD in the sense of completeness and consistency.

A DD is intended to describe, complete and exactly, the data of an IS. The DD contains metadata, ie, information about data, and data environments. With a suitable structuration and adequate tools, a DD becomes an invaluable source of documentation (Jesus et al, 1992).

Current commercial tools for DD are not able to handle any interesting cooperative features. On the contrary, the functionalities of our tool (Aguayo et al, 1994) must cover the following aspects:

- Standardization of the identifiers in every sub-IS. From the description of data, a rule-based system can infer a list of (possible) valid identifiers for these data.
- Integration of different user's views. Every user defines his own view, and the system must infer a consistent, meaningful transformation diagram.
- Exclusivity in metainformation handling. Is necessary to restrict every sub-DD only to the data defined by its user, with a few exceptions. Precisely because of these exceptions, an intelligent cooperative system is needed.
- Cooperative and internal management of consistency problems. Current DD technology implies a lot of human work to achieve consistency. A more flexible technology would allow automatic consistency checking and, in a cooperative environment, force to a trade-off between the users involved in the conflict.
- Consistency tests for final transformation diagrams and context diagrams, in order to control and audit work progress.
- Formalization of the description of the organization, and explorative organizational analysis. From a strategic point of view, this one would be the most interesting return of an intelligent DD. Since a global DD models in a certain way inter- and intra-data flows, it is conceivable to develop a tool for simulating and analyzing modifications in the IS. Moreover, a relationship between DD and modeling modules can be defined in order to improve the quality of decision maker's activity. On the other hand, the functional primitives could be used by another methodology to obtain a second level model; not a model of the organization, but one of the knowledge of every user about the organization.
- Integrity maintenance in the DD. A discipline for total integrity assurance is easy to implement, but cumbersome to use. An intelligent tool should provide enough flexibility, without loosing the ability to detect current inconsistencies.
- User-friendly interfacing, customized interfacing and user-dependent interfacing. The interface could be seen as a device mapping conceptual or DD primitives with external or

user representations.

- Multiple data modeling.
- Automatic generation of conceptual schema in order to implement the database system, like in the TRAMIS system (Hainaut et al, 1992).

3. CONCLUSIONS

It could be said that a new approach and a way to understand this piece of software and its actual significance for large companies application development has been established.

One of the main contributions of our research has been the creation of an intelligent and cooperative DD based on the theoretical aspects explained here. As this working prototype is at its developmental stage, it does not yet cover all the aspects outlined here. However, as we have already explained, it controls a set of cooperative errors and it has an intelligent interface reconfigurable by the user, at the DD and at a graphic level.

This approach is indeed challenging, but it is also a breakthrough. We believe it could resolve a great deal of the current software crisis. The user would be the designer, always keeping in mind that such a user will need thorough knowledge of his company or organisation, knowledge not available to the other employees.

4. REFERENCES

Aguayo, A., Guevara, A., and Gálvez S., (1994), Almacenamiento de datos en un sistema de información cooperativo inteligente. In Proceedings, 4th Internatinal Conference of New IT. La Habana.

Batista, B., (1991), Data directories: research issues, in II Workshop on Intelligent and Cooperative Information System, pp. 85 88.

Cuena, J., (1992), Contribution to a unifiying view of software development, in Workshop AIFIPP92.

Downs, E., Clare, P., and Coe, I. (1988), Structured Systems Analysis and Design Method. Applications and Context, Prentice-Hall.

García, H., (1991), Data management for intelligent and cooperating information systems, in Proceedings, II Workshop on Intelligent and Cooperative Information System, pp. 104 107.

Guevara, A., (1992), Planning methodology of information system based on bottom-up and top-down strategies under a cooperative design, in Proceedings, 12th WCC IFIP92.

Guevara, A. (1993), Planning methodology of information system under cooperative design, in Proceedings, XIII Int. Conference of the Chilean Computer Science Society, pp. 527 535.

Hainaut, Cadelli, Decuyper, and Marchand (1992), Database CASE tool architecture: Principles for flexible design strategies, in Proceedings, 4th Int. Conference CAiSE92. Lectures Notes in Computer Science No. 593, Springer-Verlag, pp. 187 207.

Harmon, P., and King, D., (1985), Expert Systems: Artificial Intelligence in Business, Wiley Press.

Kaplan, S.M., Carroll, A.M., and MacGregor, K.J., (1992), Flexible active support for collaborative work with conversation builder, in Proceedings, ACM 1992 Conference on CSCW, ACM Press.

Jesus L., and Carapua R. (1992), Automatic generation of documentation for IS, in Proceedings, 4th Int. Conference CAiSE92. Lectures Notes in Computer Science No. 593, Springer-Verlag, pp. 48 64.

Newell A. (1982), The knowledge level, in Artificial Intelligence, Vol 18, pp. 87 127.

Shepherd, A., Mayer N., and Kuchinsky, A., (1990), Strudel. An extensible electronic conversation toolkit, in Proceedings, CSCW90, pp. 93 104.

Steels, L., (1992), Reusability and configuration of application by non programmers, in Workshop AIFIPP92.

THE PURPOSE OF INFORMATION SYSTEMS IN THE COMPUTER INTEGRATED PRODUCTION SYSTEMS

Anton Čižman

University of Maribor, School of Organizational sciences, 6400 Kranj, Prešernova 11, Slovenia

1. INTRODUCTION

Today many manufacturers are beginning to introduce new technology on the base of a structured approach. The first reason is that, using computers on a completely different way of operating may be more efficient. The second reason is a tremendous advantage supposed to be gained by enabling computers in different sectors of the company to communicate and exchange information. For that purpose, it is most important for each company to shape a strategy for Computer Integrated Production (CIPS) at the earliest possible stage. CIPS is a strategy, incorporating computers in a sense to link the existing technology and human recourses to optimise business activity. CIPS includes in fact each activity within the organization such as production, marketing, sales, engineering, materials, finance and personnel from tendering to post-delivery service.

The efficiency of the implementation CIPS methodology, the incorporation the technology and a computer aided IS into existing factories depends in a great part on the generic factory model. On the base of the analysis of a considering manufacturing system, a generic architecture for real-time production control can be progressively developed and implemented.

Data management and network communications capabilities must be introduced to facilitate data transfer between the various production, engineering an business activities. It is vital for any CIPS implementation that a communication's network exists giving all areas of organization access the same data. Functional areas, the flow of material and the flow of information in a manufacturing enterprise shown in the Figure 1. are similar for many factories.

Effective CIPS requires the software in order to provide a control to the automated processes and facilities, and to generate information to support operational, tactical and strategic decision making. Considering to this facts, IS are needed to summarise and integrate a vast quantities of data generated by computerised production systems (CPS).

The system architecture is founded on the classic hierarchical, or pyramid- shaped, command/feedback control structure that is common of many complex organizations. This approach ensures that size, functionality, and complexity of individual control modules is limited. The unique features of the hierarchy include: the number of generic nature of the control levels, (2)the amount of the real-time computations, sensory processing and data transfer performed at each level and (3)modules ensuring that a human can comprehend and interact with them.

The amount of the real-time computations by a control module depends on the planning horizon i.e., the period of the time over which the module is responsible for planning and updating the local goals. To achieve these goals, each level control module executes one of the following common tasks: *decomposes* the current input command from its supervisor into procedures to be executed at that level, *issues* the subcommands to one or more subordinate modules, and *sends* status feedback to the supervisor. The decomposition process is performed until to the lowest level, where sequence of co-ordinated primitive actions is generated which control shop floor equipment. These principles, based on the closed loop control, support adaptive nature at each level of the control and represent the background for the various CIPS applications.

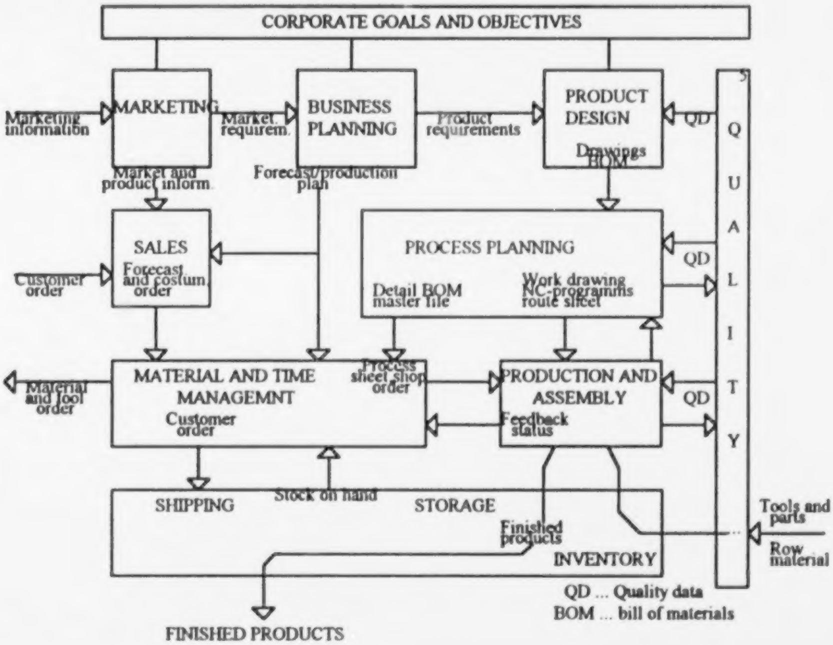

CORPORATE GOALS AND OBJECTIVES
Marketing information
MARKETING
Market. requirem.
BUSINESS PLANNING
Product requirements
PRODUCT DESIGN
Drawings BOM
QD
Q U A L I T Y
Market and product inform.
Forecast/production plan
SALES
Forecast and costum. order
Customer order
PROCESS PLANNING
QD
Detail BOM master file
Work drawing NC-programms route sheet
Material and tool order
MATERIAL AND TIME MANAGEMNT
Customer order
Process sheet shop order
PRODUCTION AND ASSEMBLY
QD
Feedback status
Stock on hand
SHIPPING
STORAGE
Finished products
INVENTORY
Tools and parts
Row material
QD ... Quality data
BOM ... bill of materials
FINISHED PRODUCTS

Figure 1. Functional diagram of the manufacture enterprise

The number of the levels in the control hierarchy depends on the complexity of the production system. Traditionally, for example small batch production systems, the organization structure is partitioned into five levels: a *facility (plant)*, a *shop (centre)*, a *cell (work centre)*, a *workstation (unit)* and an *equipment (machinery/process)*. This control structure enables partitioning the functions and the data bases needed to meet the production requirements.

2. INFORMATION SYSTEMS IN THE CONTEXT OF THE CIPS

The hierarchical production control model and functional requirements, represented in previous chapter, generate the needs to apply appropriate IS for control purposes and for managerial decision making at various levels throughout the organization. In this respect, IS associated with different functional areas, such as an inventory control, physical distribution, cost accounting, purchasing, marketing etc., need to be integrated to allow timely decision making by foremen, middle managers and senior level executives.

Conceptually, integration in a production system warrants consideration from several perspectives. One of them is vertical integration which allows the capability of the four major

693 BEST COPY AVAILABLE

types of IS: electronic data processing (DP), management information system (MIS), decision support systems (DSS) and expert systems (ES), to be accessed by managers within the organization. In this paper the attention is given to MIS to illustrate the significance of applications the of IS into the CIPS system. (see Figure 2).

MIS ensures the capability to perform highly structured tasks and limited support for decision making. MIS output is generally intended for middle managers at the shop or plant level within the factory. The important difference between production-oriented MIS and EDP systems is in the managerial orientation of the MIS output and the degree of integration across the production areas.

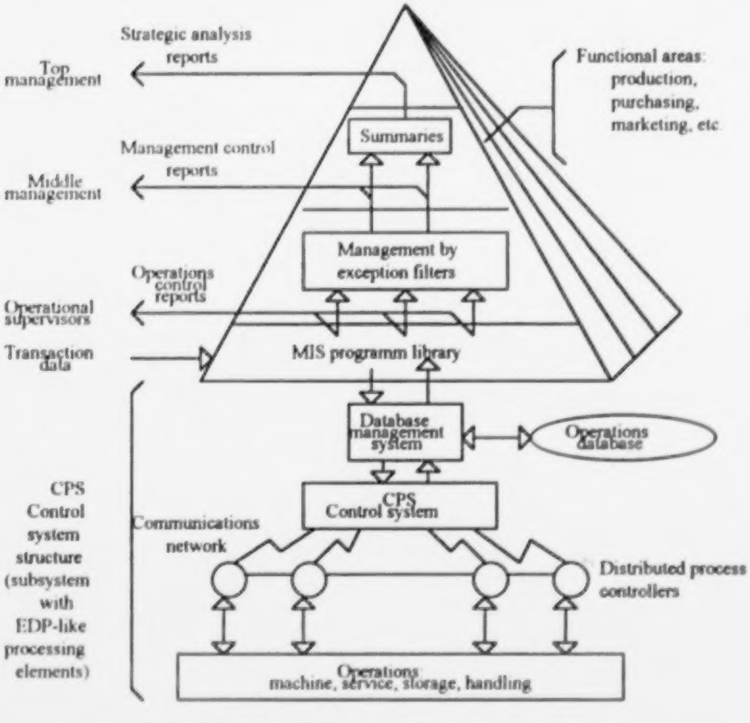

Strategic analysis reports
Top management
Functional areas: production, purchasing, marketing, etc.
Summaries
Management control reports
Middle management
Management by exception filters
Operations control reports
Operational supervisors
Transaction data
MIS programm library
Database management system
Operations database
CPS Control system structure (subsystem with EDP-like processing elements)
CPS Control system
Communications network
Distributed process controllers
Operations: machine, service, storage, handling

Figure 2. The structure of the MIS component in CIPS

The integration of management science approaches, statistical analyses methods, and operations research techniques into automated production systems is relevant feature that enables a direct support for manufacturing management decision making. To identify
preferred choices between alternative actions in a CIPS environment, various models can be embedded in a MIS. Determine the optimal product mix with linear programming, manufacturing resource planning and production process simulation are few examples of models.

3. CONCLUSIONS

A standard factory model must include all the necessary functional control, data flow and interface issues. Such model should be based on fundamental scientific principles and be partitioned into sub modules that can be comprehensive by system developers. The hierarchical production model will be the foundation for CIPS implementation.

Information systems (IS) are important components in a CIPS design methodology. From practical perspectives, systems integration provides decision makers at every level within manufacturing organization with the capability to increase the productivity. Ready access to analytical methods and variety of databases enables modelling and quantitative analysis. The purpose of IS is to provide the linkage between the production control technology and the organization system that can be defined as decision making structure.

The integration of various IS capability is paramount to achieving a truly comprehensive CIPS system including CPS and MIS needed for manufacturing management planning and operational control functions. True integration in CIPS requires a combination of four basic types of IS (EDP, MIS, DSS and ES) to provide qulitative information required for managerial decision making at varius levels throughout the production organization.

A suitable tool of IS development would be the SAS System-Integrated Tool (SAS Institute, Inc.) including prototype applications for Decision Support, Statistical Analysis, Statistical Quality Control, Report Writing and Graphics that on a large scale support the functions involved in MIS or DSS information system.

REFERENCES

Bullers W.I., Reid Jr. and R.A., (1990), Toward a Comprehensive Conceptual Framework for Computer Integrated Manufacturing, Information & Management 18, pp. 57-67, North Holland.

Čižman A., (1992), A management information system in the context of CIM, Proceedings of International Conference on Organization and Information Systems, pp.583-594, Bled, Slovenia.

Jones A.W.T. and McLean C., (1986), A Proposed Hierachical Control Model for Automated Manufactoring Systems, Journal of Manufactoring Systems, Vol.5, No.1, pp.15-25.

Kochan A. and Cowan D., (1986), Implementing CIM, Springer-Verlag, Berlin-Heidelberg-New York-Tokyo.

SAS Institute Inc., (1991), Introducing the SAS System, Version 6, First Edition, Cary, NC: SAS Institute Inc..

Shogan A. W., (1988), Management Science, Prentice-Hall, New Jersey.

SEEKING THE ACTUAL REASONS FOR THE "NEW PARADIGM" IN THE AREA OF IS ANALYSIS

Václav Řepa

Prague University of Economics, W.Churchill sq. 4, 130 00 Praha 3, Czech Republic
E-mail: REPA@VSE.CZ

1. INTRODUCTION

Lately the very attractive and frequented terms are the "object orientation" and the "change of paradigm" in the area of information systems analysis. After the object revolution in the area of programming the areas of analysis and design are now attached by these thoughts. Unfortunately, not all of the principles, techniques and methods for the object-oriented programming are fully implementable also in the area of analysis and design. So the old truth that analysis is something quite different from the programming is gradually fading away. The aim of this paper is to indicate the *actual reasons* for the paradigm change using analysis of the *general problems* of the "structured methods" of IS analysis and design.

By the term "structured approach to IS development" I mean various data-oriented as well as function-oriented methods of IS development. One of the most complete concepts of Structured methods is, for example, the Yourdon structured method [YOUR-1] which includes both data and functional models on the conceptual level of the IS development. Yourdon method recognizes the need of integration of these two (data and function-oriented) approaches, until recently regarded as substantially different.

2. GENERAL CHARACTERISTICS OF THE "STRUCTURED APPROACH" IN IS DEVELOPMENT

As the name of this approach indicates it is based on the separation of different areas of interest:

- Data and processes are taken into consideration separately
- Hierarchical abstractions are used for separating the abstract (high level) concepts and their relationships from the subordinated (more detailed) ones
- Modelling as the key principle enables the developer abstract view on the general characteristics of the information system deprived of their particular shape which is complicated by various non-essential aspects
- Building information system gradually on three different levels - conceptual, technological and implementational (the "Three Architectures" principle) reflects the effort to separate three different, relatively independent problem areas: essential

concept based on the model of reality, technological design and implementational shape of the system

The main reason for such separation is simplification of the problems which have to be solved. Even if the system is relatively small there are too many aspects in too complex relationships to describe them simply (i.e. clearly). And there are too many relatively simple problems in too complex relationships to solve them at the same time. Separation seems to be the vital condition for the mental control of a problem.

Some of the general characteristics of the main forms of separation mentioned above will prove as the source of problems leading to the need to change the paradigm of IS analysis.

3. THE MOST IMPORTANT PROBLEMS AND CONTRADICTIONS INSIDE THE "STRUCTURED APPROACH" TO IS DEVELOPMENT

This section summarizes the problems and contradictions in Structured methods of IS development. There are three main problems:

- integration of the data and the functional models
- integration of the functional and the control models
- transition from the conceptual to the technological model.

All of these problems follow from the separation and all lead to the same solution - to revise and change the Separation paradigm.

Problem of integration of the data and the functional models manifests itself in various forms. The most visible form is the inability to decide what should be the starting point of IS analysis - the data or the function analysis. When the development process starts with the analysis of functions then the idea about stored data structures follows strictly from the need of the functions. Such data structures are real contrary to the normalized data model - one data element typically occurs in several data structures. Later correction of the data structures according to the rules of normalization leads to the situation when each function needs to contact number of data stores to gather required data. This fact dramatically increases complexity of the function model. Therefore Yourdon method so strongly emphasizes the need of clear description of relationships between data stores and data entities in the data dictionary according to the balancing rules. But, unfortunately, it does not tell how to secure the clarity of this description in the process of the parallel functional and data models development. On the other hand the development process starting with the analysis of data leads to the idea that functions are just supporting algorithms, providing the inputs and the outputs to/from the database. So there is no way to ensure the fact that the function structure naturally reflects the real world processes. The algorithms which reflect the relationships of the real world events are scattered into particular operations of the input routines and the database integrity constraints. But, unfortunately, this approach does not tell how to find these constraints and operations in the real world.

Problem of integration of the functional and the control models has the same roots. Using the tools offered by Yourdon method, the developer is able to clearly describe hierarchical function structure and the relationships of the functions on particular levels of hierarchy (i.e. particular data flow diagrams). On the other hand he (or she) is able to clearly describe particular control algorithms using state transition diagrams. But how to describe which way the functions and the control algorithms match together? Trying to

697

describe it the developer very soon feels the need of accepting and producing data flows by control processes but, unfortunately, this is not legal in Yourdon method. Control process can accept and produce only control flows in Yourdon method. There is a good reason for such limitation: when the method would accept data flows as input and output to/from control process then the question is: What is the natural difference between function and control process? The difference between function and control processes is very significant in structured methods. This is because the function is taken here as top-down decomposable element while the control process always must be described as elementary process. So the nature of control processes is very similar to the nature of data entities (they also are not top-down decomposable). Viewing this fact from the object (i.e. not structured) point of view we can see the control processes in the system as the expression of the objects behaviour - their life history. The structured methods are not able to reach this point of view because of their need to take the data and the process models of IS into consideration separately.

The third main problem area of the structured methods - *transition from the conceptual to the technological model* - we can regard as the consequence of the previous two problems. On the technology level of the IS development the integration of the data and the functional models as well as the integration of the functional and the control models must be realized unconditionally. With poor description of these facts on the conceptual level it is really a hard task to ensure this integration on the technology level. And the structured methods do not offer the explicit way to ensure it.

4. CONCLUSIONS - WHAT THE SCOPE OF IS ANALYSIS ACTUALLY REQUIRES FROM THE "NEW PARADIGM"

The duty of the progress always should be removing the negative features of the old state. So the main contribution of any paradigm change should be a new approach which overcomes the limits and problems given by the old paradigm. Consequently, to ensure the progress during the paradigm change the exact analysis of these problems and limits is necessary.

This section summarizes the actual requirements for the new approach to the IS development which follow from the analysis of the problems and contradictions of the Structured methods stated in preceding section. The new approach should:

- *bring to life the method of conceptual analysis which fully accepts the natural unity of data and processes.*

To achieve this goal it is necessary to develop a tool - diagram of the conceptual model of the IS. The basic element of such diagram is an object. By the term "object" I mean the unity of data (attributes of the real-world object) and process (life history of this object). Each object in the conceptual model represents an object of the real world. Diagram of the conceptual model must also be able to describe the conceptual relationships between the objects. Each relationship between two particular objects represents the causal dependence of their life cycles. It is realized via sending the information about essential events from one object to the other. Data of this relationship then represent the attributes of the event. Michael Jackson whose method JSD [JACK-1] can be regarded as object - oriented distinguishes between two types of object dependencies:

- passive dependence realized as sending the information about the event from one object to the other and
- active dependence realized as watching the attributes of one object by the other.

The difference between stored and sent data expressed in Jackson's classification seems to be technology dependent. But much more likely it reflects natural differences between two essential types of the object relationships.

- *give the possibility of hierarchical decomposition (of the collectivization type) of the system functions inside the new method of conceptual analysis.*

Although generalization is natural type of the abstraction of objects (it is expressed by the principle of inheritance) there is the actual need of hierarchical decomposition of the system functions as well. The reason for such request is the same as the reason for the collectivization in the Structured methods - it seems to be the vital condition for having the problem under control. By the way, the complexity of the model is one of the main problems in the current object-oriented methods of IS analysis. The difference between two types of system processes must be recognized in order to apply the collectivization in object-oriented methods. For those processes which are modelling essential behaviour of the objects (their life history) the generalization type of the abstraction is natural. But the essential model usually contains also other processes - those which provide the outputs and inputs of data from and to the system. And these processes do not model the real-world objects behaviour, nevertheless they depend on it. So they also seem to be the part of the essential model. The natural type of abstraction to be used for those processes is the collectivization. Of course, it is the question whether such output - input processes should be regarded only as a part of the technological model.

- *ensure easy and not contradictional transition from the conceptual to the technology level of IS design.*

The transition from the conceptual to the technology level of IS design is a big problem in the structured approach to IS development. It requires complete revision of conceptual analysis results and complete change of the approach to the system design. Significant contribution of the new paradigm should be its ability to solve these problems. Also in current object-oriented methods of IS analysis these problems still exist or, better, they are replaced by the new ones. Evolution of the methods for IS development brought to life several significant conceptions and principles known under the names as "modular programming", "programming in the large", "composite design", "information hiding" etc. Not all of them are fully accepted by the object-oriented methods. But many of the principles found in those conceptions are valid in general.

REFERENCES:

[JACK-1] Jackson, M.A.: System Development, Prentice-Hall Inc., Englewood Cliffs, NJ, 1982.
[YOUR-1] Yourdon, E.: Modern Structured Analysis, Prentice-Hall Inc., Englewood Cliffs, NJ, 1989.

HUMANIZATION OF ORGANZATION AND INFORMATION SYSTEMS

Duško Uršič

University of Maribor, School of Business and Economics
Razlagova 14, 62000 Maribor, Slovenia

1. INTRODUCTION

In order to attain a competitive advantage, the organization depends more an more heavily on the human being and his/her creativity; we call this process humanization, and find it revolutionary in modern times, and at the same time heavily dependent on the development of organization and on its support by information systems. In this respect, the essence of the information systems is not the development and/or application of computers, but rather feeding information in the form of knowledge, findings, understanding, decisions, etc. The actual humanization of organization will assert itself once we succeed in linking the individual, group and organizational intelects with the efficiency of formal and informal processes in the organization. The information systems play a special role which results from permanently new requirements for a competitive control over oneself and one's organizational and market environment.

2. HUMANIZATION OF ORGANIZATION

Contemporary research of humanization of organization points out the relation between cognition and knowledge, real problem and action, first of all, as a relation between phenomenology (= understanding) and methodology (= doing besides understanding). The humanization of organization is based on systems thinking. Therefore it is multidimenzional, because it is grounded in philosophy of science, theory and practice of society and of organization (Krohn et al., 1990).

The basic dimensions of the presented definition of the humanization of the organization are therefore the following ones:

1) ontology, the reality and the concept of its research,
2) epistemology, which presents a definition of the research area, thus the definition of the system researched,
3) human nature, which presents a man's (non)rational behavior as impacted by culture, experiences and his own proper selection of thought and action, and
4) methodology, which presents selected variables, measured in a specific way in conditions of probabilities, and deals with them in a more or less creative way (Waring, 1989).

The identity of the informations system must reflect the contents of all four attributes cited above.

The beginning of the defining the humanization of organization is therefore primarily not a problem of techniques used, but of why and what to do for its humanization. Contemporary findings about the humanization of organization defend the thesis that the absolute objectivity in research does not exist, therefore the one objective truth about information systems does not exist either.

According to this statement, the concept of humanization of organization should consider at least two viewpoints as follow:

1) the researcher's relevance about the concept is discovered in a more or less well-known way, which mostly represents a quantitative dimension of systems thinking,
2) the researcher's relevance about concept is discovered in a well-unknown way, which mostly represents a qualitative dimension of systems thinking.

Contemporary systems theories have explained the relations between the quantitative and qualitative viewpoints of humanization of organization at least in connection with cognition about entropy, feedback, requisite variety and selforganization (Ur{i˘, 1994). The viewpoints mentioned are holistic and therefore logical only then, if they form a consistent entity which represents a specific change of researcher's view of himself, with other words, of his own humanization of his way of thinking.

Without a permanent up-dating of the information systems and other supports to creativity, a kind of a research trap may show up and transform creativity and the process of humanization to ideology with dogmas and with no room for facts opposing the well-known ones. This can be in the domain of organization and information systems as well. Humanization of organization, based on the trap mentioned is therefore fictious and counts only under the conditions of a closed system kind of researcher's consideration of himself as well as the of subject of research, including the information systems.

We conclude from that, that the humanization of organization should take place as a humanization of researcher's (systemic) thinking and therefore his/her research approach, as a humanization of the object researched, including an information systems, and as a humanization of their environment.

The awareness of the organizational humanization and the process of its surfacing, therefore, decisively impact also the conceptualization of information systems which provide for a basis for people to understand themselves and their environment, to conceptualize their decision making process, to implement it, and to assess and judge its effects.

3. TWO BASIC ISSUES OF AN ORGANIZATION - THE BASIS FOR A SIMPLER DEFINING OF AN INFORMATION SYSTEMS

P. 1 showed that our knowledge about the organizational humanization supports our attempt to control our ideas and deeds concerning the organizational humanization which we cannot do without a satisfactory information. When doing this, we unavoidably come across the simplification process which enables us to achieve the attributes mentioned earlier. Without simplification, the problem is namely to broad for us to understand and handle successfully. We need to keep the identity of the organizational system and to add particularities of the concrete case under research and action, at the same time. A critical judgement of a general concept of information systems and its comparison to their special and individual reflection, is a result of avoiding the exaggerated (!) technical, hard-systemic approach which may lead to losing the identity, by the humanization of organization. Once we make this, we can no longer avoid a deliberate simplification of information systems adapted to the specific purposes of research and action. We are forced into it by criteria with which we assess the competitive advantages of our own or somebody else's action. Now, we will again become interested, first of all, in the background causes urging the simplification of an information system's conceptualization.

Thinking about information systems in organization is much simpler if we first ask what does an organization result from, rather than what is it. In this respect there is no essential difference between dealing with an information system and dealing with an organization. On the contrary. From a systems science viewpoint this are problems which have some general common characteristics. Hence, one's creativity and capability to express oneself are important for the mutual relation between an ('absolute') finding and knowledge, and a finding which is accompanied by random events, on one hand, and the human values which, in principle, change permanently in their qualitaty and quantity, on the other hand. We actually have to do with two problems along with defining of information systems in organizations:
(1) to find out what and why are the starting points of defining of information systems in organization,
(2) to find out real components to be defined on the basis of the finding about the first problem.

The interdependence of both problems is obvious. The way of their solving depends on a logical differentiation of the individual experiences (which exposes multidimensionality), and on the difference between expectations and reality. This is the real problem of the general and the specific definition of information systems in organization because the different subjective and objective viewpoints are to be considered. The awareness that there exist differences is, hence, unavoidable when defining the information systems: it supports conceiving of preconditions for a better understanding in managing of diversity. The ever new differences between perceptions, expectations and reality lead to ever new findings about the information systems in organization. Broader the capability to define an information system in organization, easier the perception of the problem situations which are, as a rule, subjected to different levels of stohasticity in the relation between reality and expectations. Since the basic reason of defining of information systems in organization lies in the finding how to deal with complexity in the real life, a rapid changing of the circumstances and conditions in fields of the new theoretical and practical knowledge puts two (research) issues:
(1) what is stochasticity like when defining an information system in organization,
(2) what is stochasticity like when solving a problem situation on the basis of a defined information system in organization.

Both issues trigger four types of (new) problems linked with defining of information systems in organization (Ackoff, 1978):

(1) the problem of learning: we know in principle what is the problem of information systems, but we do not know how to deal with it,
(2) the problem of operation: we know what is the problem of information systems and how to deal with it, but only in principle,
(3) the problem of compromising: we know how to deal with the problem of information systems but we are not sure, and hence we do not know in which (alternative) way to deal with the problem,
(3) the problem of inspiration which does not have its basis in a 'clear' finding of what the information system is and how to deal with it, but in an 'unclear, fuzzy' one.

We are now facing a problem situation that questions whether or not it makes sense to define an information system in an organization at all. The answer cannot be (only) a uniform one, a 'yes or no', it would not help a lot. We would be better off if we found out the causes and the consequences. The causes stem from the finding that something exists and is dealt with as an information system in an organization. The consequences stem from the finding that a perception of what is an information systems in organization may be linked with a given real situation and one can find out what actually influences its changing.

4. CONCLUSION

The modern competitiveness depends more and more on the ability to consciously control many processes which decisively define the individual and his/her organizational environments. Solution suggested by findings on the humanization of organization and information system tend to apply a systems approach to the topic discussed. Why? The answer is simple: a holistic consideration of all influential factors has proved to be more competitive. The transition of a partial to a wholistic dealing with information systems, though, does not depend on a change of techniques to be applied. In the forefront is, first of all, the issue of the humanization of organization. How succesful will it be, depends on many factors. Among them, of course, the most important one is the individual and his/her own humanization as well as the humanization of the organization. Findings about them could enable also a more satisfactory approach to an attempt to understand the starting points for dealing with information systems.

5. REFERENCES

Ackoff, R.,(1978), "The Art of Problem Solving", Wiley, New York

Krohn, W., Kuppers, G., Novotny H.(eds.),(1990), *in* "Selforganization, Portrait of a Scientific Revolution", Kluwer Academic Publishers, Dordrecht

Uršič, D.,(1994), Evolution of Quality and Quality of Systems Thinking, *Systems Research*, Vol. 11, No. 1, pp. 87-100

Waring, A.,(1989), "Systems Methods for Managers" Blackwell Scientific Publications, Oxford

Socio-Technical Systems Design Re-visited

Christopher C. Wills

Kingston University, School of Information Systems, Penrhyn Rd, Kingston, Surrey
KT1 2EE, U.K.

Introduction

Any system which involves a human input should be regarded as a human activity system[1]. A human activity system consists of both technical and social components. The Socio-technical systems (STS) approach to the design of computer information systems involves attempting to concurrently design and optimise, both the social and technical components of the system. This is done by encouraging users to participate in the design of the system.

The author is in the early stages of reviewing the way in which computer based information systems are designed and implemented, in a large pharmaceutical company. This paper represents work in progress. Initial indications are that the organisation seems, through a process not dissimilar to neo-Darwinism, to have adopted a socio-technical approach to systems design.

Almost all information systems design methodologies have developed out of the mathematically based traditions of engineering design. This tradition has tended to emphasise the need to optimise the technical component of the system. One of the assumptions of engineering design has been that once the system (machines and processes) has been designed and built, the people can be slotted into the system[2]. Hitherto, there has been little widespread consideration of designing systems that "fit" the people who will work with them, or indeed of other organisational perspectives .

Mumford and Henshall (1983) argued that, "The training given to systems analysts is, to say the least, very much biased towards computer systems design, data manipulation and organisation techniques. It recognises the human element of the system in an almost apologetic and certainly mechanistic fashion....This is not to say that systems analysts do not think about the human factor; they do, but they do not have the methods, tools and training at their disposal which would allow them to design systems that satisfy both the technical system and the social system requisites."

Although the STS approach has been understood in the UK for some time, it has not been widely adopted as a design methodology by the computing industry. However a recent conference organised by the British Computer Society[3] seems to signal a renewed interest in the approach.

---

[1] See Checkland (1981)

[2] See (for a critique of "Hard" systems perspectives) Bjorn-Anderson, (1980)

[3] Technical Change and Organisational Design, October 1993 BCS London

There is a significant body of evidence gathered over a number of years, largely by Enid Mumford, Emeritus Professor of Manchester University, which suggests that the STS design approach to the design of computer information systems may be superior in some aspects to other IS design approaches. STS seems to elicit more accurate system specifications, and creates an environment in which users find it easier to relate to the systems which sit at the centre of their work.

As Mason and Willcocks (1994) point out, "The principle of the socio-technical approach to implementation is the observation that a successful computer system is the outcome of the right mix of hardware, software, people and procedures. If any of these are designed in isolation, the other elements will suffer and the overall efficiency of the application will be adversely affected. The proponents of the socio-technical view believe that most computer application are designed with too much emphasis on the technical aspects. Too often in the past, technically superior computer systems have failed in practice, because the human factors involved have been overlooked".

The realisation that the effective operation of any system which involves an interrelation between people, processes, and machines is determined by the proficiency with which these factors interact, has been long understood.

The theoretical and empirical antecedents upon which the STS approach to the design of computer based information systems has been built, is that of the work of Trist and Bamforth (1951), and the Tavistock Institute of Human Relations, London. This early understanding that the neither the technical, nor the social components of a human activity system, should be optimised to the point of detriment to either, has paved the way for a better understanding of the problem of designing computer information systems.

The central theme of Enid Mumford's work, has been to incorporate the outcome of the Tavistock's findings into an approach for the design of computer information systems. This approach emphasises the need to involve users in the design of the system.

Mumford identifies three levels of involvement, or user participation. They are, in ascending order of the extent of user involvement in the design process; consensus participation, where all the users affected by the system are included in membership of the design group; representative participation, where only representatives of the affected users are part of the design team; and consultative participation, where the users or their representatives are consulted at various stages of the design process, but not closely involved in participating in the design of the system.

The extent to which the users can participate in the design process is determined to some extent by the willingness of the host organisation to devolve the responsibility for design. To a greater extent, it is determined by the number of users affected by the system. It is more difficult to extend direct (consensus) participation in the design process to a large group of users than it is to a small group.

Direct participation by a large group in unwieldy and impractical. The logistics involved in a large group of people meeting regularly in a decision making forum are complex and costly. Representative participation in the design of the system is also potentially problematic, for much the same reasons that representative political democracy can be problematic. Consultative participation may not be participative enough to involve the users to the point where they will feel that they own both the process, and the system that results from it.

It is clear however, that although the participative approach to design is not without problems, even at the consultative level, it still engenders a feeling of ownership of the system amongst the users.

Engendering a feeling of ownership is important. Participation by users in the process of change results in them feeling less threatened by change. A reduced perception of threat is likely to reduce the users resistance to change.

As important is the deep-rooted emotional feeling of ownership all of us experience when we embody our intellect and creativity in the creation of something external to us. Users who feel that they have a personal stake in the system will be highly supportive of it.

The other outcome of user participation is that the knowledge elicitation process is made more effective: "Firstly, it is a fact that the people with the greatest knowledge of the existing formal work system, and the people with the most knowledge of the informal system, are the people who are who are responsible for their operation. They therefore have the greatest potential to successfully design a system which overcomes the present systems shortcomings"[4].

Notwithstanding these practical justifications for using an STS approach for information systems design, there is also a moral philosophical dimension to the argument. Clearly, people should be involved in the design of the systems with which they work. They should work in situations where they rewarded equally well in both intrinsic and extrinsic terms.

Mumford's STS methodology, ETHICS (Effective Technical and Human Implementation of Computer-based systems), has fifteen stages, the core method involves the following -:

Forming the design group, be it on a consensual, representative, or consultative basis.

Appointing an "expert" analyst designer, who is attached to the group, to act in the role of "facilitator", helping the group reach conclusions about what is feasible, by offering expert advice, rather than by directing the group to a particular conclusion.

The design group then identifies the primary social and technical requirements of the future system. The group creates a list of social alternatives, and a list of technical alternatives which will contribute to attaining the primary social and technical objectives. These two lists will contain those features that should be embodied in the future system, in rank order of desirability.

Social alternatives will focus on those features which provide the best social solution to the problem, (typically, those features which improve the quality of working life of those who will work with the system, or are affected in some way by it). These features will be considered in terms of what is possible **(social possibilities)**, what is desirable **(social needs),** and what is undesirable or prohibitive **(social constraints)**.

Technical alternatives will be concerned with those features which will provide the greatest technical improvement to the problem, and as with social alternatives, will be considered in terms of what is possible **(technical possibilities)**, what is desirable or prohibitive **(technical needs)** and **(technical constraints)**.
Comparisons are then made between the social and technical alternatives, and a list containing those social and technical alternatives which are not mutually exclusive is compiled.

---

[4] Mumford E. & Henshall D. (1983) pp.120

These potentially **available solutions** are now evaluated in terms of costs/benefits, and the extent to which they satisfy both the primary social and technical objectives, the outcome of this process being the highest ranking STS solution.

Latterly, Mumford's work has been used as a base for other STS approaches[5], and indeed incorporated with other approaches into a flexible methodology called Multiview.

Multiview contains five stages which integrate a number of existing methodologies. Stage one centres on the construction of "Rich Pictures" set out by Checkland (1981), stage two on function and entity models. stage three on STS, stages four and five are self explanatory.

- Analysis of human activity systems
- Information modelling
- Analysis and design of the Socio-technical system
- Design of the human-computer interface
- Design of the technical subsystem

In the preface of their book, Avison and Wood-Harper (1990) argue that "The philosophy behind the third stage is that people have a basic right to control their own destinies and that if they are allowed to participate in the analysis and design of the systems that they will be using, then implementation, acceptance and operation of the system will be enhanced".

The development of Multiview , with its emphasis on human factors, is clearly a move in the direction of the application STS. If, as seems likely, STS based approaches become more widely used, one consequence will be that the analyst-designers of the near future will have to undergo a much broader based training than hitherto. Such training will have to incorporate tuition in aspects of social science such as industrial psychology, and industrial sociology, as much as it includes training in computer science.

It follows from this that the current emphasis placed by designers, and by their clients, on the technical "fix" will also change. It is difficult to predict where this potentially ensuing paradigm shift in emphasis on the perspective from which the design process is viewed will lead us; perhaps to a new and different understanding of what constitutes an appropriate application of technology in the work environment. Cooley (1987) argues that "As we design technological systems, we are in fact designing sets of social relationships, and as we question those social relationships and attempt to design systems differently, we are then beginning to challenge, in a fundamental way, power structures in society".

References

Avison, D. & Wood-Harper, A. (1990) "Multiview: An Exploration in Information Systems Development" Blackwell, Oxford

Bjorn-Anderson, N. (ed.) (1980) "The Human side of information processing" IAG, North Holland

Checkland, P. (1981) "Systems thinking, systems practice" Wiley, New York

Cooley, M. (1987) "Architect or Bee?" Hogarth Press, London pp.180

---

[5] see for example HUSAT

HUSAT Research Centre, Loughborough University (1988) "Human factors guidelines for the design of computer-based systems" Ministry of Defence Procurement Executive/ Dept of Trade & Industry, London

Mason, D. & Willcocks, L (1994) "Systems Analysis, Systems Design" Alfred Waller, Henley-on-Thames pp.271

Mumford, E. & Henshall, D. (1983) "Designing Participatively" Manchester Business School, Manchester

Trist, E. & Bamforth, K. (1951) Some social and psychological consequences of the longwall method of coal getting, in "Human Relations", 4, 1951, pp3-38

Wood-Harper, A., Antill, L. & Avison, D. (1985) "Information Systems Definition: The Multiview Approach" Blackwell, Oxford

INFORMATION SYSTEMS FOR LIBRARIANS AND USERS OF SCIENTIFIC LIBRARIES

by ANNAMARIA CAMPANILE

Central Library of the National Research Council, Piazzale Aldo Moro n. 7, 00185 Roma. (Italy)

Abstract

The term information technology stands for the sector concerned with information processing and transmission systems and related technologies.

This sector plays a role of strategic importance in the economic and social development, in that it conditions the competitiveness of production systems and the efficiency of services.

It is with this end in view that libraries and technologies have established an increasingly close relationship in the rendering of services, since it is just by the application of certain technologies that libraries can reach higher standards of efficiency and effectiveness by deriving resources and opportunities from them to increase and enhance services to users.

It has become necessary, therefore, to examine and manage the service of libraries and the role of librarians in the light of electronic communication: this marks the passage from traditional libraries to virtual libraries.

Traditional libraries were organized to accomplish the function of information storage and documentation assigned to them from the remote past, in that libraries were the depositories of man's knowledge. Today, on the contrary, we talk about electronic or digital libraries and global or virtual libraries, which collect all information sources with the purpose of making the information searched, more or less remote, freely and easily accessible and available to users.

But how have conservative librarians reacted to these changes and by what instruments do we intend to face so "timely" informed users?

The development of information systems has been the keystone to the needs of virtual libraries, making available automatic systems, the use of networks, the organization of qualified services such as library-to-library lending and document delivery, to achieve more qualified goals such as computer-aided publishing and electronic document transmission.

In Italy, under the pressure of the request for information made by its users (researchers, university teachers, professional or university students), the CNR Central Library has laid the foundations for developping and renewing the services rendered, as a result of the imposing requests for information. With this end in view, the library's primary activities have become "made to measure for the user", in the sense that, apart from routine activities concerned with cataloguing, greater incentive has been given to the activities connected with the

BEST COPY AVAILABLE

location of all existing information sources, thus laying emphasis on the natural "addressee" of the whole library system, the user.

Obviously, these changes have been gradual and programmed, always taking into account the dynamic character of the library system which, right or wrong, keeps up with the progress of science: in fact, the new disciplines originated in this century (cybernetics, informatics, ergonomics) have brought a wind of innovation so as to make it open to greater prospects.

These changes have involved particular efforts on the part of librarians and end users.

As far as librarians are concerned, the application of new technologies and new methods has led to an enrichment of this professional figure as a result of the acquisition of new tasks and responsabilites, first of all the capability of communicating with users, and willingness and openness to technological changes.

The CNR Central Library, for the purpose of interacting with its users and in line with the changes which the information systems are undergoing, has laid special emphasis on the adaptation of its structures, in particular those connected with the management and spread of information, since the idea of an only-storage library is absolutely obsolete.

These changes have resulted in:

1) the electronic catalog of monographs and periodicals, with local and remote consultation network;
2) the organization of information distribution through document delivery;
3) the consultation of more than 60 data banks on CD-ROMs for document delivery;
4) on-line inquiry of data banks, scientific and non scientific;
5) the creation of an interface with other libraries, information centers, and other information sources;
6) the databases of the national scientific libraries, with the resulting organization of a common catalog, for shared data entries and remote inquiries.

The development and management of documents which are very often of the electronic type has given rise to the need, on the part of the Central Library, to make its users autonomous, in consideration of the greater possibilities of customized search offered by the on-line catalog compared to the paper catalog. Furthermore, the same prospects offered by the virtual library, by the media library intended as the whole of documents of rare nature, electronic and on paper, have found new applications for the users who, in the process of spread and circulation of information, are increasingly demanding and differentiated.

Hence comes the need to prepare our professional librarians to accomplish the function of "communicators" for the access of users to the automated procedures.

With this end in view, training courses have been organized on the consultation of electronic catalogs, search on CD-ROMs and data banks, methodological techniques of information search, not neglecting the traditional manual search compared to the new technique offered by computers.

The impact of the new technologies on the end user has thus determined the opportunities and the limits of the user's training, for a more effective use of the library service, rich in technologies managed in self - service for search on catalogs and on-line databases, reproduction and remote transmission of documents, etc.

The techniques of instruction of users have been planned and inserted at various stages of implementation:

- guided tours for the presentation of the services offered by the Central Library;
- lectures;
- training and refresher courses;
- vocational training;
- instructions for the consultation of data banks;
- scientific seminars;
- round tables;
- scientific meetings intended to enable end users and librarians to exchange ideas and opinions.

These actions, carried out programmatically, with suitable instruments and resources, have enabled librarians, the managers of the information service, to acquire exceptional skills. This has resulted in the greater willingness of users to understand and adapt themselves, facing problems in the search and retrieval of data, with an active and dynamic participation in the activity which has increased the demand of users and enhances the service.

On the other hand, the librarian himself, the mediator of information, does not wait for the requests of users to improve his activities, but he tries to locate the needs, first, in order to give the right solutions in advance.

Since the objective of the Central Library is the access to documents (no longer possession), the activities that distinguish it today are connected with the location and classification sources, with the constant aim of instructing users on how to search not only in the library's catalogs and repertoires, but also in catalogs and remote data banks through communication networks that are offered to them for an effective and timely service.

References

Cardinal, M., (1993), Du contenant au contenu, Documentation et Bibliothèques, vol.39, n. 4.

Faedda, G., ed., (1984), I servizi della biblioteca e l'utente. Atti del 32° Congresso nazionale AIB. Roma, AIB, 1987.

Johnson, M.L., (1987), Information broker: a carrer in scientific and technical information service in Science and Technology, vol. 7, pp. 3-9.

Lippman, M.J., (1993), The library as information producer, in: Journal of Documentation, vol. 49, n. 1.

Campanile, A.M., (1992), Informatizzazione dei servizi documentari e di biblioteca nel settore scientifico-tecnico. Esperienze della Biblioteca Centrale del CNR, Bari, Area di Ricerca.

A CONCEPT OF PROJECT INFORMATION SPACE OF THE ES-GDSS TYPE

Andrzej Zaliwski, Marian Kuraś

Cracow Academy of Economics, Department of Computer Science,
27 Rakowicka St., 31-510 Kraków, POLAND,

1. INTRODUCTION

The paper presents the general outline of a software tool conceptualized and developed in anticipation of changes in IS design. The system ES-GDSS type (deSanctis and Gallupe, 87) (VIB-CASAD) should help to overcome fundamental difficulties in design encountered by the authors using ISSAD methodology (Kuraś, Sarga, Zaliwski, 92). Changes in the organizations - deep and quick in Poland - accelerated learning of management rules and how to use the information technology (Kuraś and Zaliwski, 93). Anticipating changes of approach to IS design, authors have worked on the construction of the methodology and computer-aid for 8 years. ISSAD methodology is expected:

- to contain the whole scope of work performed during IS development;
- to be an alternative guide through methods, techniques and tools available in the literature (Avison, 92);
- to offer a convention of graphic description of IS structures which supports the structuring of problems;
- to impel standard solutions in scope of options typical for the soft system approach (Avison, 92);
- to offer a computer-aid to most labor-consuming activities in analytical work;
- to contain knowledge about methods, techniques and tools (handbook, contextual "help", and error diagnostic program) available for the user;
- to transform itself into a knowledge based system, supporting co-operation between user and analyst;
- to be tested before being supplemented with support tool.

The ISSAD methodology has been applied in six IS projects (Kuraś et al. 90; Kuraś et all. 92). It is regularly used with student's projects. The general outline of a computer-aided tool was proposed and developed (Kuraś, 92; Zaliwski, Kuraś, 93). It consisted of the idea of Virtual Intelligent Board (VIB).

2. VIB-CASAD - ES-GDSS Information Space

The construction of CASE tool is an undertaking within a borderland of three fields: management, technology, and social environment. Each of the actors use different language and have different association scope characteristic for one's occupation.

Common disturbances in communication between diverse occupational groups are not only an excellent occasions to make jokes but they are major reasons for failure of many expensive software projects. Both, the user and designer (Fig. 1) conceptualize models reflecting their ideas about the modernized system. It is necessary that these models overlap as a result of user and analyst exchange of information. This is the only way to avoid faults arising from mutual misunderstanding.

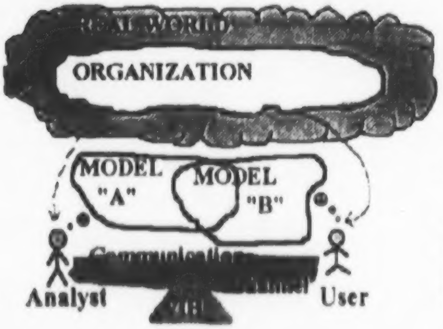


Figure 1. VIB-CASAD as a communication . medium.

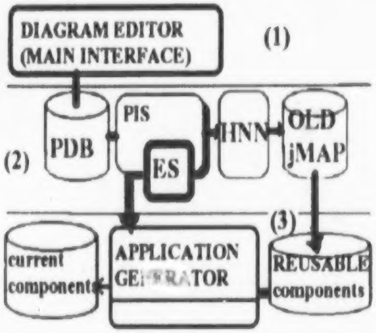


Figure 2. Structure of VIB-CASAD system.

There are many other barriers to IS construction and development: difficulties arising from a system multiplicity, organizational and human barriers i.e. lack of understanding of organizational strategic plans and needs. This happens particularly when users do not agree about goals and objectives of the modernized IS. Additionally, there is a great redundancy of documents generated during the design process. As the implementation of system comes to its finish, the amount of documentation rapidly increases. The formal translation among different representations of software and supporting documentation does not practically exist (Jaworski, 89-93). It is very difficult to assure the coherence of all the documents. As a result we get several parallel descriptions of the same problems' different aspects. It is difficult to find out a satisfactory partial solution (reusable specification for example) in one project and to apply it to another.

The proposed system (VIB-CASAD) is firstly quasi-intelligent communication medium ('secret sharer') which enables analyst to overcome typical disturbances in the design process; Secondly it is a tool for gathering, storing, and retrieving 'knowledge' about IS during design session.

A full concept of 'Project Information Space' (PIS) of the ES-GDSS type (deSanctis, Gallupe, 87) incorporate a user and an analyst into considered system.

The main purposes of this system are:

- to provide a framework and support in discourse between the user and the designer;
- to find out contradictions in specifications of a designed system;
- to gather facts during a session and preserve their integrity;

- to suggest what data might still be needed,
- to produce IS specifications using knowledge bases.

Three distinguished VIB-CASAD layers are to perform following functions (Fig. 2):

- **Design Layer** is a communications tool between the user and the designer. Main element of this layer is a specialized graphic editor supporting the first stage of analysis i.e. gathering data about the considered IS. Editor operates interactively to support communication between system, and user and analyst is based on diagrams interactively developed according to the rules of ISSAD methodology. Putting any object or flow on a diagram generates the equivalent object in PDB (Project Data Base).
- **Transformation Layer** consists of mechanisms to support design process:
  - transforming descriptions of an organization into PDB object data base. All the descriptions gathered on the Design Layer are stored and made accessible to redraw diagrams and for analysis;
  - retrieving solutions from previous projects similar to these currently undertaken;
  - offering notation for not completed constructions and the preservation of rejected solutions to avoid *information loss*;
  - maintaining previous versions of specifications.
- **Application Layer** contains application generator using information from previous layers. The generator uses the definition of classes selected from the general base of "Reusable Component Library" (RCL) on the basis of similarities to the old projects filtered through jMAPs.

According to Jaworski (89-93) we recognize the software development as a process of gathering knowledge. Already known information about considered IS is gathered in PIS knowledge base. The progress is achieved by acquiring further information (from topmost layer) and incorporating it into the structure of the knowledge base or by the processing of already known information. Representation of knowledge about the IS project at every stage is supported by the jMAPs notational technology (Jaworski, 91-93). The main advantage of using jMAPs is that they support documentation of the whole process of IS development on a single *sheet* structure image of developed system. In the next step we can use pattern recognition techniques to compare *image* (or its parts) with *images* of other systems stored in 'Old jMAPs' database. F*uzzy & associative search* (HNN) is a module of pattern recognition. Application of jMAPs to represent knowledge about IS enables the developers to produce simple and uniform structure of the PIS. Functionally PIS[1] is an equivalent of *repository* in other commercial systems.

Information acquired from topmost layer is stored in PIS. PIS is a multidimensional and multilayer model of current IS project state reflecting a part of reality. This model may be treated as a set of hypotheses and apart from other actions as a subject of research carried out by an expert system. Its knowledge bases comprise:

- a model of outer environment i.e. information about outer conditions in which an organization is active and which have or may have impact on its activity;
- a model of an inner environment i.e. facts about an organization (a specification of information requirements and client's demands).

Additionally the knowledge and PDB bases consist of:

- limits and boundary conditions of a project;
- methodology guidelines;

---

[1] First outline concept of *Project Information Space* originates from Webster (87).

· diagram syntax rules (on the top layer);
· vocabulary - defined and extended by a user .

The VIB-CASAD system will support analyst along the process of requirements definition, IS specification and development providing them with specific knowledge necessary to solve problems at every stage of a project.

3. CONCLUSIONS

The proposed VIB-CASAD is a complex tool to store and to make the knowledge accessible about the current IS. It uses the jMAP notational technology as the representation of this knowledge. Applications of jMAPs to the fuzzy & associative search allow retrieving previous solutions, similar solutions in previous projects, and changing from one notation to another. Descriptions of projects previously carried out with the support of this tool will be gathered in knowledge bases. This will help testing of fuzzy & associative search. Now a prototype is elaborated of basic functions of the VIB-CASAD. The real danger to the project (partially implemented) is lack of sufficient funding. The main system modules will soon be completed. The VIB general outline additionally requires elaboration of philosophy of design team group work using an intelligent board.

4. REFERENCES

Avison, D.E., Golder, P.A., Shah, H.U. , (1992), A tool kit for soft systems methodology, *in*: "The Impact of Computer Supported Technologies on Information Systems Development", IFIP Transactions A-8, Kendal, K.E. et al. (eds.)., Elsevier Science Publishers B.V.

Jaworski, W. M., Cummings, T., (1991), Normalization and optimization: using infoMaps as an inspection and program processing tool, Canadian Conference on Electrical and Computer Engineering, Quebec City, September .

Jaworski, W.M., Deslauriers B., (1990), Software process, maintenance, products: Software development environment based on infoMaps, Control and Electrical Eng. Conference, Ottawa.

Jaworski, W.M., Zaliwski, A., Kuraś, M., (1993), InfoMaps: an alternative method of knowledge gathering, V Wiosenna Szkola PTI, Szczyrk-Katowice (in Polish).

deSanctis, G., Gallupe, R. B., (1987), A foundation for the study of Group Decision Support Systems, *Management Science*, Vol. 33, No. 5, May.

Kuraś, M., (1989), On the scope of IS modernization, *Zeszyty Naukowe AE Kraków*, Nr. 286.

Kuraś, M., Sarga, D., Węgrzynowski, A., (1990), Requirements specification with users participation, *Zeszyty Naukowe AE Kraków*, Nr. 330. (in Polish).

Kuraś, M., (1992), General outline of computer aided analysis and design, *Zeszyty Naukowe AE Kraków*, Nr. 366. (in Polish).

Kuraś, M., Sarga, D., Zaliwski, A., (1992), A methodology for system planing and development, Research Rep. 29/ISEI/7/91/S, Katedra Informatyki, Akademia Ekonomiczna Kraków. (in Polish).

Kuraś, M., Zaliwski, A., (1993), Communication aspects of IS development, V Szkola PTI, Szczyrk-Katowice. (in Polish).

Maiden, N.A., Sutcliffe, A.G., (1992), Exploiting reusable specifications through analogy, *Comm. of the ACM*, Vol. 35, No. 4, Apr.

Sarga, D., (1993), Function description of a hypothetical organization, AE Kraków. (in Polish).

Webster, D. E., (1987), Mapping the design representation terrain, Technical Report STP-093-87, MCC, Software Technology Program., July.

Zaliwski, A., Kuraś, M., (1993), CASAD - A new approach to IS modernization, V Wiosenna Szkola PTI, Szczyrk-Katowice. (in Polish).

INFORMATION RESOURCES INTEGRATION PROBLEMS

Adam Nowicki and Cezary Stępniak

Academy of Economics, Economic Computer Sciences Institute, Wroclaw
ul. Komandorska 118/120, Poland

Abstract

Decision problems which are solved in modern organizations require information resources. They often require possibility to access to databases belonged to different kinds of information systems. In the way, there is the important problem of information resources integration origin from different systems and belong to different owners. The requirements of integration process realization often step over possibility of one organization and absorb plenty of them, which come from the same environment.

There are a lot of research problems, which have to be solved before and during realization these processes. The principal aim of the paper is to show: information needs as a premise of information resources integration process.

Information resources questions will be present at the paper as a research problem. The questions will be considered in support of empirical researches, which was made in Polish conditions.

1. INTRODUCTION

The word 'integration' has became popular in different kinds of human activities recently. It appears, that in Management Information Systems (MIS) domain, this notion can find its application.

The aim of this paper is to pay attention on information needs problems in the overorganizational environment as a premise of information resources integration at MIS. The question has many motifs as investigation problem. So in this paper we want to say about:

- information needs in overorganizational environment as a potential information resources,
- proposed types of information resources connection.

We touch some chosen questions from this domain and its carrying back in Polish business practice. The carrying back will achieve on basis researches, which was made in Polish in 1992-94.

Before we will talk about chosen problems of the domain, first we would like to explain the essence of information resources integration in MIS. We recognize the information resources integration process as a join of two or more information systems into one unit system, in that way:

- there is possibility to exchange information resources between compounded systems,
- there are uniform roles of system resources using,
- there is separated decision center, which supervise system functioning,
- users belong to different organizations.

Modern telecommunication technology with adequate software tools allow to use information resources, which belong for other organizations. But usually it is possible only with special agreement, which the information proprietor can give. In the effect, only chosen group of users can receive needed information. These processes are rather examples of *exchanging* information, then *integration*.

We can talk about information system integration, on condition conscious activity of information system proprietors and users for realization united, integrated system. It is easy to see, that the realization of these information system types depends on the coordination activity of potential users, who coming from different kinds of organizations. The aim of the functioning system should be automatization of information processes which are *between organizations*. So we call overorganizational environment the place, where the integration process should be realized.

Processes of integrated systems creation depend upon integration of chosen information system elements (information resources) of different organizations into one unit. It is possible, when organizations voluntarily make accessible chosen parts of their information resources to use at the integrated system.

2. INTERORGANIZATIONAL INFORMATION NEEDS AND RELATIONS.

We treat information needs as a very important premise of information resources integration. We made investigation to define information relations and needs in overorganizational environment and to research potential attitudes of organizational management to realize of the system which could satisfy and support it.

The investigation was realized in 15 Polish enterprises (they have 100 -2000 employees). We intended to define information relations between the enterprises and environment. We used utilitarianism criterion during analyze of received materials for that. This criterion is based on the verification of relations noticed at are chosen enterprise using information relations at others enterprises. As the result, we select 45 types of links 'environment - enterprise" and 63 "enterprise - environment" (we appreciate the same or similar links as an one type).

We suppose, that there are significantly more kinds of links. We know, that there are types of links, which were not notice during investigation (it was some kinds of reason). The other problem is that number of these links is not constant. It is known, that information relations can change itself on contents, flows, directions or numbers.

We define three groups of the researched relation using this criterion. They are:

- the first group of information relations follows from running business activity,
- the second group is connected with data collection about market economical situation, it is used for business planing, simulation and preparing business strategy,

- the third group follow from lawful norms and prescriptions, which are established by government or different kinds of public offices and institutions.

The investigation of potential attitudes of organizations management was realized by using questionary. The questionary was sent to 40 firm (potential users of planning system). It appeared, that about 85% organization is interested in use of these systems, but only 22.5% offer to take part in building it. So the results show, that Polish enterprises are interested in systems, which can integrate information resources belonged to their environment, but majority of them prefer to wait for ready product ,

Requirements are another problem which was standing up by potential users of the system. The satisfying of the requirements are often the condition for cooperating with the planning system. Examples of requirements are in table 1.

Table 1. Users requirement.

Requirement	Percent
Full information services	42.5 %
Relative less costs	37.5 %
Data quality	32.5 %
Data exchanging	32.5 %
Easy data access	25.0 %
Influence for system develop directions	12.5 %
Another	7.5 %

3. INTEGRATION TYPES.

After analyzing information needs of organizations and their potential attitudes to the planning system, we can mark out three types (levels) of integration processes:

- formal integration,
- autonomous integration,
- total integration.

Formal integration consist in joining of suitable elements directly (in this way putting systems, which was integrated). The aim of this kind of integration is developing common activity. Simultaneously, each element (putting system) realizes own separated tasks and it can be disconnected from the system at any moment. We can notice effects of this kind of integration as follow:

1. Integrated system (SZ) consist from **n** elements:

(1) $SZ = \{e_1, e_2, \dots, e_n\}$

2. Each element (putting system) realizes only its own separated tasks:

(2) $\forall\, e_i = \{z_{i1}, z_{i2}, \dots, z_{iqi}\}$

qi - is an index of i-element tasks.

3. Realizing tasks set (signify **Z**) of integrated system is equal sum tasks of all putting system..

. (3) $Z = e_1 \cup e_2 \cup \dots \cup e_n$

Right side of equation is called *an own tasks set of putting systems* and it is signify **F**. So the equation (4) is true:

(4) $F = e_1 \cup e_2 \cup ... \cup e_n$

Concluding the consideration, if realizing tasks set (**Z**) is equal an own tasks set of putting systems (**F**), we can say, that it is common connection integration (equal 5).

(5) $Z = F$

Autonomous integration is when putting systems realize added tasks of the integrated system (it is *an integration effect* and is signified **T**) beside its own tasks set and they can be disconnected from the system at any moment. The integrated system realizes following set of task(equal 6):

(6) $Z = F \cup T$

Total integration consist in indissoluble joining putting system into one integrated system. The elements realize new tasks, which are important for whole system and they were impossible to do before integration. After integration, elements lose its independence and its disconnection cause disintegration of the whole system. Realizing tasks set of integrated system is an effect of integration (old tasks of putting systems are usually included into new functions of integrated system).

(7) $Z = T$

4. SUMMARY.

In this paper, we talk about information resources integration problems in overorganizational environment. We want to pay attention, to that making decision processes require information from organization environment, too. So we want discuss about the possibility to realize information system (in the paper named planning system), which could integrate information resources needed for managers in making decision process. In Poland researches on this field are in the theoretical stadium, but we think that business needs will extort realization these kind of information systems.

5. REFERENCES

Awad E.M (1988).: Management Information Systems. Concepts, Structure, and Applications. The Benjamin/Cummings Publishing Company, Inc. 1988.

Curtis G. (1989): Business Information Systems. Analysis, Design and Practice. Addison-Wesley Publishing Company: London 1989.

Earl M.J.(1991): Information technology and strategic advantage. In "Information Technology Management. Some topics. Economist Hogeschool Limburg: Diepenbeek - Belgium.

Flakiewicz W. (1990): Informacyjne systemy zarządzania. PWE: Warszawa.

Hall V.J., Mosevich J.W. (1988): Information Systems Analysis with an Introduction to Fourth-Generation Technologies. Prentice-Hall Canada Inc., Scarborough, Ontario 1988.

Kendall K.E., Kendall J.E.(1988): System Analysis and Design. Prentice Hall. Englewood Cliffs.

Kisielnicki J. (1993): Informatyczna infrastruktura zarządzania. Wydawnictwo Naukowe PWN, Warszawa.

Nowicki A.,Stępniak C.,Zielińska M., Bytniewski A.(1992): Determination of Information Needs in Process of Creating of Large Management Information System. The Third International Conference on Information Systems Developers Workbench. University of Gdańsk. Sopot.

Nowicki A., Stępniak C. (1993): Utylitarność regionalnych baz informacji gospodarczych. Zesz. Nauk. AE, Wroclaw.

Ochman J. (1992): Integracja w systemach informatycznych zarządzania. PWE, Warszawa.

AN INTEGRATED, FULLY OBJECT-ORIENTED SOFTWARE DEVELOPMENT METHOD

Benet Campderrich

Dept. d'Enginyeria Informàtica, Universitat Rovira i Virgili, Tarragona, Catalonia, Spain

1. INTRODUCTION

MOOI is a software development method that is being developed as the object of a joint research project of the University Rovira i Virgili and the University of Girona. The goal of the project is to develop a method that covers all the life cycle of the software, from the description of the information flows within the organisation to the program design, using essentially the same most basic o-o concepts in all the life cycle phases. The main underlying ideas are exposed and an outline of the different phases is presented.

A lot of object-oriented (o-o) software development methods have been put forward so far; some of them are oriented to just analysis or design, and other cover both, as those of Coad and Yourdon (1991) and (1991a), Rumbaugh, Blaha and Premerlani (1991), Jacobson (1992) and Booch (1994) among other. A technique for the description of the organisations by means of o-o concepts has been presented in Graham (1993).

The following usual object-orientation concepts will be used: object, attribute, class, subclass (specialisation/generalisation, inheritance), service or method and message, and the aggregation operation, which defines complex objects as assemblages of simpler ones. We think it is not necessary to introduce them here, as they are well-known.

Some features of MOOI are the following:

- the information interchanges within the organisation are defined in o-o terms: the organisational units are seen as objects that perform services requested by means of messages
- the contents of a message is considered a complex object, so it can be defined by means of o-o operations in terms of its component objects, which are data values
- the classes on which the design is based are inferred from the data contained in the messages
- an *object derivation language*, that defines a complex object in terms of other objects, is used for different purposes: to describe the structure of a complex object, to specify the contents and format of an output and to describe services. One such language is the one described in Campderrich (1994), that is based on the BLOOM o-o data model from Castellanos, Saltor and García-Solaco (1991).

2. DESCRIPTION OF THE LIFE CYCLE PHASES

2.1. Organisation dynamics modelling

The only aspects of the organisation dynamics that concern us are the information exchanges. The organisational units (departments of any level and individual posts) are considered as objects of a particular sort, which we call *bureaus*. A bureau often is the only object in its class, but it is not always so; for example, all the sales offices of a firm can be considered to be the bureaus of one class, or the different possible purchase requesters in a firm (perhaps all the departments of it, no matter their respective function) can be seen as the bureaus of one class in a purchase system. A bureau can be a complex object resulting from aggregation of component bureaus: for example, a personnel department can consist of several sections in charge of payroll, social security, taxes, etc. Two special bureaus are devoted to information storage and retrieval: ***Database*** , in the case of computer-stored information, and *Archives* for manual storage and retrieval.

The activity of the bureaus consists in services (manual, automated, and mixed), which can be initiated either at the request from another bureau through a message or at the bureau's own initiative. A service can be split up into simpler services performed by the bureau or by its component bureaus.

In this context, a message consists of the identifier of the sending bureau, the identifier of the receiving bureau, the requested service and sometimes data to be used to perform the service. We say that these last data constitute the *document* of the message; a document is seen in MOOI as a complex object resulting from the aggregation of its component data values.

During this phase the bureau classes are identified, along with their services and the messages associated as input or output to these services. Neither the services nor the documents are still described in detail. A definition language is used, and bureau-message diagrams can be drawn optionally at different possible levels of detail.

2.2. Document analysis: referenced class identification

The goal of this phase is to infer additional classes from the data contained in the documents; these classes will be the basis for the specification of services (*bureau services analysis*, in the terminology of this paper) and the definition of databases. The classes so defined play a role similar to that of the classes inferred from the nouns in a descriptive text in other methods.

The data that constitute the documents are the attributes of objects (we call them *referenced objects*, belonging to *referenced classes*). Documents either make reference to these objects (customers, products, suppliers, persons, etc.) or are the objects themselves (orders, invoices, etc.).The referenced classes can be initially inferred on the basis of the genitives used in the designation of data (we call this the *genitive rule*); for example, if there exists a data item known as 'supplier-name', a 'supplier' class can be fairly reasonably postulated, and 'supplier-name' will be an attribute of it. This rule could be partially automated, and the correctness of the referenced class inference depends strongly on how accurately the document data have been named.

Sometimes a referenced class corresponds to a bureau, as if in the preceding example there were also a 'supplier' bureau identified in the previous phase; but the supplier bureau and the supplier referenced class would be two different classes because they have different services, for instance, 'accept order' the bureau and 'create supplier' the referenced class.

This phase begins by defining the documents *contents* (the exact document *format*, in the case of reports, is specified in a subsequent phase) in terms of the component data, and this data as well; then referenced classes are synthesised by aggregating the data items with a common genitive; the data names with no identifiable genitives are kept apart until the next phase.

2.3. Construction of the referenced class structure

The goal of this phase is to obtain a class structure for the set of referenced classes which includes the aggregation and specialisation/generalisation structures and the dependences between these classes. This will certainly require additional information on the organisation. In this phase the initial referenced classes from the preceding phase are reviewed, the data names with no genitives are attributed to suitable classes, additional subclasses are identified and the relationships between classes are defined. The class structure obtained is not necessarily complete, because new subclasses and services can be identified on analysing the bureaus' services in the next phase.

States of the objects and *transitions* between these states can also be defined in this phase. The concepts of state and transition between states are used in many methods; in our context, states can be seen as a special sort of subclasses, whose main characteristic is that their objects can individually move from one subclass to another. For example, the class 'supplier orders' can be specialised in the states 'orders sent to the supplier', 'orders accepted by the supplier', 'orders delivered', 'orders invoiced', etc.; on the other hand, two subclasses such as 'male employees' and 'female employees' can hardly be considered states of the class 'employees'. Another characteristic of a specialisation of a class into states is that each object in the class belongs to exactly one state; nevertheless, two or more independent specialisations into states can conceivably be defined on the same class. *Substates* can be distinguished within one state as the result of further specialisation. Like subclasses in general, states can have attributes and/or services of their own, i.e. not shared by all the objects in the class; following with the previous example, the orders in the state 'orders invoiced' can have a method such as 'pay invoice' and an attribute such as 'invoice no.', that don't exist for orders in other states.

A diagram representing the referenced class structure can be drawn with conventional o-o notations like Booch's or Coad and Yourdon's.

3.4. Bureau services analysis

Usually the services of the bureaus create and transform documents[1], what means aggregating or deriving complex objects or changing the state of them, and sometimes also creating non-complex objects (elementary data values) to be aggregated into the documents. So bureau services can be described, at least partially, by means of a suitable object derivation language.

In this phase, at least all automated and mixed services are specified in detail. Mixed services must be broken up into manual and automated ones. As a result of the services analysis additional classes, subclasses (and particularly states) will be defined sometimes. The analysis of the manual services is not necessary for the software development, but it must not be neglected if one wants to analyse and design a complete information system, and not just software.

3.5. Persistent classes definition (database definition)

This phase has two steps: first, an o-o database design is created, and afterwards this design is translated to the terms of the database management system to be used. The o-o database design consists of persistent classes, which are those referenced classes whose objects have to be stored because they are expected to be used later on (as opposed to documents, that are supposed, as far as this method is concerned, to be created when the corresponding message is sent and destroyed when it is received). The persistent classes are the referenced classes inferred from the data contained in the documents of the messages going to or coming from the special bureau called Database, as mentioned before. A specific class diagram can be drawn if desired, but it is possible to use the referenced classes diagram instead, just marking on it the persistent classes.

As every persistent object has to be at least created and deleted by the Database bureau at the request of other bureaus, new methods can be defined in this phase, which are assigned to suitable bureau classes, sometimes new as well.

---

[1] An exception would be what can be termed *control services*, which deal with messages containing no stricto sensu document; these messages correspond to control flows in structured methods.

3.6. Process, program and report format specification

In this phase the automated services are grouped into processes, which are subsequently divided into programs, and the format of reports is specified in detail in terms of the object derivation language (the report contents has been described in a previous phase as a document). The starting point is the output of the previous bureau services analysis phase. The program specification is highly dependent on the programming language to be used, as probably in all design methods, and has not been elaborated yet.

4. CONCLUSIONS

The MOOI method uses repeatedly the basic concepts about objects in its successive phases to describe different aspects of the software to be developed and of the information system in which this software is to be used. Organisational units, message contents, referenced entities and their representation in a database, all are objects, whose attributes and behaviour are described in object-oriented terms. The information generated in one phase complements, and sometimes corrects, the information coming from the preceding ones. We think that this approach based on a single, very flexible concept used in different ways makes possible to reach a good level of both simplicity and smooth phase-to-phase transition in the method proposed; for the same reason, the method seems fairly easy to learn, and to automate, at least partially.

As remarked in the introduction to this paper, the version of MOOI presented here is a draft one, which is currently being reviewed and extended. The development of some simple CASE tools and of a program design step (and subsequently program generation), probably for C++, are planned also.

6. BIBLIOGRAPHY

Booch, G. (1994): "Object-Oriented Analysis and Design with Applications", 2d.ed. Benjamin Cummings.

Campderrich, B. (1994): "A Closed, Data-Model Integrated, Object-Oriented Query And View Definition Language". Paper submitted.

Castellanos, M., Saltor, F., García-Solaco, M. (1991): "The Development of Semantic Concepts in BLOOM Using an Object-Oriented Metamodel". Report LSI-91-22, U.P.C., Barcelona, Spain.

Coad, P. and Yourdon, E. (1991): "Object-Oriented Analysis", 2d. ed. Yourdon Press/Prentice Hall.

Coad, P. and Yourdon, E. (1991a): "Object-Oriented Design". Yourdon Press/Prentice Hall.

Graham, I. (1993): "Object-Oriented Methods" 2d. ed., p318-323. Addison-Wesley.

Jacobson, I., et al (1992): "Object-Oriented Software Engineering: A Use Case Driven Approach". Addison-Wesley.

Rumbaugh, J., Blaha, M. and Premerlani, W. (1991): "Object-Oriented Modelling and Design". Prentice-Hall.

USING EXPERT SYSTEMS IN CAPITAL INVESTMENT

Petr Wolf, Jana Drastíková

Technical University of Ostrava, Faculty of Economics, Sokolská 33, 701 21 Ostrava 1, Czech Republic

Abstract:

This paper gives more insights into some questions concerning using expert systems in capital investment. The paper describes a model for decison - making process for increasing capacity of the production equipment.

1. INTRODUCTION

Parallelly with changes in the political systems of the Czech Republic, essential changes take place in its economy. A shift from a centrally controlled and planned economy appears to be the most crucial change. In addition, transition to a market economy has resulted in an economy opened to the world. The production has undergone demonopolization, expecially manufacturing industries. This will enable a gradual privatization and development of private enterprises. All these changes are associated with the adoption and introduction of modern methods of management and also with the utilization of computer techniques.

2. INFORMATION SYSTEMS

These days, a price of information plays a dominant role on the market. That is why the present is often described as "the era of information technologies". Information technologies and related technical equipment more and more frequently turn into a strategic tool which will enable one to improve the efficiency of a firm, to get a big jump on competitors, to implement new methods of management, to create new conditions for free enterprise to simplify paperwork processing, to get rid of inefficient work and the like. Thus the information technologies become a part of the system of management.
Currently, orientation in the field of information systems (with a view to both software and hardware) is shifting from the field of methodological questions of information systems creation to the questions of strategic significance of informatics for the development of economic organizations. At present, one can hardly imagine a company functioning effectively without computing.

The industry focused on application software development meets at present wtih serious problems associated with the solution of tasks. For example, poor software quality, shortage of qualified software designers, insufficient communication between designers and users and management.

3. EXPERT SYSTEMS AND MANAGEMENT

Expert systems are one of the most important technologies which are used for manager's decision supporting.
Management is a process in which goals are reached by using the sources (people, money, energy, material, space, time). Success in the manager's work is often to be measured as the difference between output and input.
This is indicated as the productivity of the organization where

$$\text{Productivity} = \frac{\text{output (production, services)}}{\text{input (sources)}}$$

In consideration of the mounting frequency of the changes and uncertainty of the surrounding world higher demands are put on decision making - process and management.

Table 1. illustrates the changes of the main factors influencing the manager's decision - making process. The results indicate unambiguously the increasing burdensomeness of the decision - making process.

Table 1. The influence of the select factors on the decision - making process

FACTOR	TREND	RESULT
Technology	Increasing	More alternatives
Information/computers	Increasing	Great quantity, problem of timely gaining, validity of information
Size of organization	Increasing	Increasing price of the "arising" faults
Competition	Increasing	More uncertainty for future
International markets	Fluctuating	New markets gaining is difficult
Political stability	Fluctuating	Especially in the areas of the new perspective markets (previous U.S.S.R). Hesitating in investment (central and Eastern Europe)
Consumption	Increasing	Looking for new sources, ecological problems

Table 2. illustrates combination of the type of problems and management level in decison - making process.

The decision making process depends on the level of the complexity of the solving problem.

Table 2. Combination of the type problems and management level in decision - making process

MANAGEMENT LEVEL				
TYPE PROBLEM	OPERATION MANAGEMENT	MANAGER CONTROLLING	STRATEGIC PLANNING	DEMANDS SUPPORT
STRUCTURED PROBLEMS	Orders, Invoices	Budget analysis, management of the part of production	Financial management	Managing informa-tion system, models of the operation research
SEMISTRUC-TURED PROBLEMS	Planning of production, Stock control	Development of credit budgeting	Firm development	Decision support system
NON-STRUC-TURED PROBLEMS	Approving of the loans, evaluating of investment purpose	Managing, hardware buying, choice of a new manager	Development of the new technology, social planning, setting of the firm strategic goals, development of the market, business plan	Decision support systems

The structured processes solve periodical problems when standard solution exists. Object of the processes, its outputs and inputs are exactly specificated.

The semistructured processes solve the problems combining standard solutions with the individual hypothesis.

The nonstructured processes solve difficult fuzzy problems, when standard solution does not exist.

4. EXPERTS SYSTEMS IN CAPITAL INVESTMENT

Expert systems are source of a new dimension during the analysis of the most important information provided to the user. Their aim is to offer advice for solving the problems in real time. You can encounter following types of the rules:

"If the capital project is to be financed, then analyze the financial markets to find the best method of financing".

"If capital is to be invested in the project then evaluate the role and position of union".

It is clear that expert systems can increase profits. The organizations using them can broaden their customer base by offering and selling better products, improving provided services, and expanding to new areas without increased staff.

One of the most interesting areas is the application of expert systems in evaluating projects for investors. This type of expert systems application is being developed at our Faculty of Economics.
Capital investment planning is a decision process for investment of a firm's capital to the projects supporting future business operations. There are several reasons that capital investment planning is of paramount importance:

* size of the projects,
* complexity of the choice,
* uncertain future,
* unreversible decisions,
* high price of the faults.

For these reasons and with regard to the top management capital investment planning contributes more then financial analytic decisions. Capital investment planning brings qualitative reasoning needed for making sure that every capital investment plan is consistent with the production and cooperative strategies.
Then it is necessary to characterize the conditions of the investor. For example:

- Investment area - engineering, technologies, ecological projects.
- Conditions.
- Structure of management.
- Products, technology on the level xy.
- Investment size.
- Market.
- Social climate.

Then follows, for example:

* year of the foundation of the enterprise,
* results of the privatization process,
* type of the enterprise,
* development of the prices of the shares,
* the key managers,
* characteristics of the key managers,
* strong and weak aspects of management,
* production capacity current,
* production capacity estimated after investment,
* investment in know-how, technology, TQM,
* share on the market,
* competition advantages,
* position of unions,
* social climate.

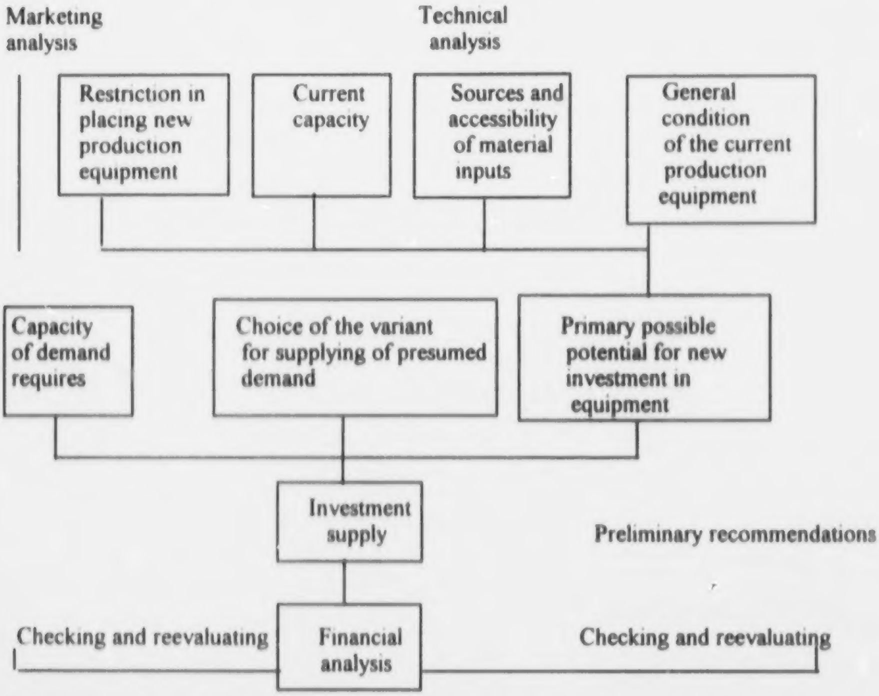
Solving of these problems is to be divided among specific groups of experts. The whole process is coordinated by a system engineer. Each of these areas requires using of the extra expert systems.
For instance - an expert will suggest an investment in the new equipment when there are following conditions:

* demand is expected to be substantially higher,
* current production capacity is limited or not able to satisfy demand,
* there are no restrictions in placing of a new equipment.

The expert tends not to recommend a large investment in equipment if:
- stable or decreasing demand is expected,
- production capacity of primary potential can satisfy an expected demand,
- existing equipment is new.

The financial consequences of these recommendations are analysed before final decision as follows on figure 1.

Figure 1: Survey of a new equipment investment to expand capacity


Marketing analysis
Technical analysis
Restriction in placing new production equipment
Current capacity
Sources and accessibility of material inputs
General condition of the current production equipment
Capacity of demand requires
Choice of the variant for supplying of presumed demand
Primary possible potential for new investment in equipment
Investment supply
Preliminary recommendations
Checking and reevaluating
Financial analysis
Checking and reevaluating

For prototype goals the system is limited by creation one of four recommendations:
- to invest in the new production equipment which will extend an existing production capacity,
- to use another possibility in organization,
- not to invest in new equipment.

A typical user's dialog covers three expert areas:

1. Capacity of the requires of demand:
   This information is gained from marketing management and from the published sales information.

2. Preliminary possibilities for investment in the new equipment:
   This information is gained from consulting with the engineers and operators who analyze operations of production possibilities. E.g. Is it an existing capacity full or limited?
3. Choice of the possibility to satisfy assumed demand.

5. CONCLUSION

The expert systems can be used to help managers in solving their decision - making process concerning capital investment. A theory and application are presented and the tables and figures which support the hypothesis are provided.

6. REFERENCES

Francett,B. (1991), "Umělá inteligence se dostává do popředí", Computerworld, Vol. 51-52, pp. 20

Burton,Ch. (1991), "Doba ledová ve světové umělé inteligenci končí, tvrdé zasvěcení", Computerworld, Vol. 51-52, pp. 24

Radding,A. (1991, "Umělá inteligence v akci", Computerworld, Vol. 51-52, pp. 22

Turban,E. (1988), Decision Support and Expert Systems, Macmillan Inc.

Sviokla,J.J. (1986), Business Implications of knowledge - Based Systems, Data Base, pp. 5-18

THREE-LEVEL CONCEPT IN THE DATA MODELLING

Jindřich Kaluža[1], Ludmila Kalužová[1]

[1] Technical University of Ostrava, Faculty of Economics, Sokolská 33, 701 21 Ostrava 1, Czech Republic

1. INTRODUCTION

Application software of information systems has been built up in many cases without special care of data model design. Data structures have often been constructed by means of intuitive approach and with the great portion of redundancy. So the attention given to data modelling should bring positive results in the process of system analysis and design. Note the term *data model* is here implemented in the sense of *data structure of the information system modelled* in contrast to the definition applied e.g. in (Date, 1990) introducing the term *schema*.

Two possible *basic approaches* to data modelling are discussed in relevant books (Jackson, 1988), (Occardi, 1992): "top-down" and "bottom-up" approach. The first one is selected as a methodological base for the method designed in following because of its implementation advantages. This approach is oriented to the specification of entities first and their attributes and functional dependencies then. On the contrary the second approach prefers the opposite procedure issuing from the attributes specification. Practical effect of such an approach is very limited especially in real conditions of a big number of attributes.

2. THREE-LEVEL CONCEPT

There is a number of attempts trying to specify the data modelling procedure originally as a *one-level process* (Teorey, 1990) which does not take into account some general, on one side, and some specific features of particular database concept (relational, hierarchical or network) on the other side. Nowadays it seems to be generally accepted (Batini, 1992) that this process could be characterized as embracing *two main phases (levels)*:

- conceptual data modelling
- logical data modelling.

Conceptual data modelling is independent on any particular database system and it is oriented to the designer-user interface. The development of conceptual modelling is connected with the general acceptance of Chen's *Entity-Relationship (E-R)* method. This method became really the base of conceptual models regardless of which modification of E-R elements was introduced (many authors since the Chen's proposal in 1970s have suggested certain complements of his basic tool set). The independence of conceptual data model is an important feature of this modelling phase because of the user-friendly communication environment. The understanding of the model design from the user side is a key factor.

Logical data modelling is fully dependent on the database system concept chosen. Compare to hierarchical and network concepts the *relational* principle is more widerly introduced in current commercial database systems environment. Truly said, relational data model is also more compatible with the E-R conceptual methodology. That's why the following considerations are based on the relational concept. Nevertheless, the object-oriented databases now frequently discussed have not brought any substantial practical result in the field of economic applications.

All presented approaches to data modelling are commencing with the entities and relationships specifications. The key problem of conceptual models is to set up the E-R structure exactly reflecting the reality modelled. Concisely said, how to recognize if the object identified reflects entity, relationship or attribute, how to begin with the modelling process?

The idea trying to answer the questions given is comprised nere in the concept of *semantic data modelling as a phase preceding the conceptual process*. The scope of semantic modelling is in the identification of *objects* expressing the nature of reality modelled. In authors' opinion this phase dedicated to the introductory analysis of the reality modelled is of key importance for the following steps. It is important to find out all relevant pieces of information, define various objects regardless of their belonging to entities, relationships or attributes in following steps of the modelling procedure. On the whole it is a *three-level concept* encompassing following three phases (levels):

- semantic data modelling
- conceptual data modelling
- logical (relational) data modelling.

The modelling process starts in semantic level in the frame of which an object structure is formed reflecting the elements of reality modelled. The model content is expressed simply verbally. It contains names of particular objects and their characteristics recognized. In the conceptual level the process continues in graphical way modelling the entities structure and relationships among them. These elements are resulting through the transformation of objects identified as the "carriers" of E-R elements. The third (logical, here relational) level carries on by further transformation to the relational structure defined by syntactic relational language. This transformation is accompanied by so-called normalizational procedures precising the internal structure of the model.

Table 1. Three-level concept characteristics

Characteristics	*Level*		
	Semantic	*Conceptual*	*Logical*
Modelling element	Object	Entity/Relationship	Relation
Language	Verbal	Graphical	Syntactic relational
Source	Input requirements	Semantic model	Conceptual model
Result	Object structure	E-R structure	Relational structure

3. THE DESIGN METHODOLOGY

Because of the limited extent of the paper the design methodology divided into *twelve consecutive steps* is in following characterized very briefly.

SEMANTIC MODELLING

1. Input data requirements identification

By means of an interviews and existing documents (forms, reports, existing data structures) analysis all input data requirements are identified and organized in written form.

2. Data objects specification

Analyzing text materials developed in previous step the *data objects* are to be recognized. Those objects are accompanied also verbally by the description of their characteristics.

3. Objects structure correction

The objects structure has to be analyzed comparing mutually particular objects and certain negative features investigated are to be corrected: synonyms and homonyms, redundancies, contradictions etc. The resulting objects structure creates the input to the conceptual modelling.

CONCEPTUAL MODELLING

1. Entities definition

The structure of entities issuing from the objects set is generated and expressed in graphical form (using rectangles). Objects could be recognized as entities when having additional characteristics in contrast to different ones being identified as attributes (see later).

2. Primary keys identification

From various characteristics accompanying the objects structure only those are selected which identify uniquely every entity. These items are called as *candidate keys*. *Primary key* is then the candidate one chosen for the entity identification. The key identifier is depicted in the upper part of the entity rectangle with the suffix # symbolizing the "key" importance. In case the part of the key is transferred from the different entity as a *foreign key*, the suffix is expressed by *.

3. Relationships definition

The relationship expresses certain connection between two or more entities (most currently between two entities so called binary relationship). Every particular relationship is characterized by two characteristics: *cardinality* and *optionality*. The first one expresses how many instances of both entities is related to each other (1:1 or 1:n or m:n). The second one shows the level of the entity participation in the relationship (optional or mandatory). Graphically the relationship is depicted as a line between entities involved (dotted for the optional participation) with doubled arrow for the cardinality on "many" side (see example in fig. 1).

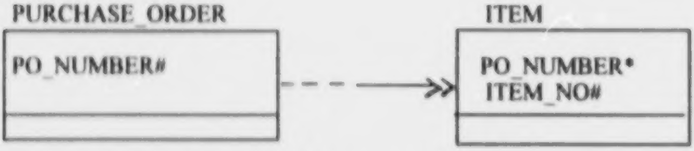


Figure 1. Graphical language of the conceptual level

4. Integration into the resulting model

All particular models produced by an individual designers are integrated into one resulting model. Also problems discovered by the integration procedure and caused by the different views of participating designers have to be solved out. They can concern the synonyms, homonyms in the entities and relationships definition, inconsistency in keys applied and cardinality and optionality defined.

LOGICAL MODELLING

1. Creation of the preliminary relations set

Preliminary relations are relations defined only by their names and candidate keys. The conceptual model is converted from graphical to the structured relational form. The rules managing this process depend on the cardinality and optionality of the relationships - see more detailed in (Jackson, 1988). Preliminary relations could look like:

PURCHASE_ORDER (PO_NUMBER#)
ITEM (PO_NUMBER*, ITEM_NO#)

2. Non-key attributes adding

The rest of attributes describing the characteristics set of every relation is added. This set fully reflects objects and characteristics involved in the semantic model not contained in the conceptual one. For example two relations shown above could look like:

PURCHASE_ORDER (PO_NUMBER#, SUPPLIER_NO, DATE, SUP_ADDRESS)
ITEM (PO_NUMBER*, ITEM_NO#, QUANTITY, PRICE)

3. The conceptual model revision

The non-key attributes adding could lead to some partial changes in the structure of entities and relationships. For example composed and multivalued attributes could be solved as new entities extending the original model. For such cases also following steps described above should be carried out.

4. Normalization procedures

All relations creating the data model have to be analyzed by means of the normalization tests. Usually the Boyce-Codd normal form (BCNF) is sufficient for practical purposes. It should be emphasized that those tests ensure the consistency of the internal parts of data model but not its general "correctness" and exactness of the reality reflection. These features of the model depend on an experience and creativity of the designer.

5. Domains specification

All domains matching the attributes defined should be specified and characterized (type, length, interval of values, permitted values etc.).

Data model designed by means of the methodology described is ready to be implemented in the environment of specific database system based on the relational principle. The design procedure is here characterized very briefly, for practical purposes it needs more detail toolset.

4. CONCLUSION

In authors' opinion the methodology described offers an effective tool for system designer to build the database structure in high quality. The implementation of the method in practical conditions forces the designer to keep more exact and systematic process in his/her work. The results obtained in students' projects in our faculty confirm the aplicability and utility of this methodology.

5. REFERENCES

Batini,C., Ceri,S., Navathe,B. (1992), Conceptual Database Design, An Entity-Relationship Approach, The Benjamin/Cummings Publishing Company, Inc., Redwood City CA

Date,C.J. (1990), An Introduction to Database Systems, Volume 1, Fifth Edition, Addison-Wesley Publishing Comp.

Jackson,G.A. (1988), Relational Database Design With Microcomputer Applications, Prentice-Hall Int. Editions, Englewood Cliffs, NJ

Kaluža,J., Kalužová,L. (1993),"The Data Models Design Methodology in the Process of Information Systems Modelling", ISSM, 1993, Silesian Polytechnic, Katowice

Occardi,V. (1992), Relational Databases, Theory and Practice, NCC Blackwell, 1992

Teorey,T. (1990), Database Modeling and Design, Morgan Kaufmann Publishers, Inc.

ISBN 86-81049-78-X

734